20/01

Logic Synthesis

RTL → gate level netlist

functions:

- how to represent

→ truth table (not practical)
(unique way of representation)
2 funct's can't have same representation

Need to remove redundancy

f^{ON} ⟷ f^{OFF} (complementary)

→ can evaluate only one

i.e. evalute minterm or maxterm (unique)

{All these are textual way of Reprecerting}

→ Graphical (Binary Decision Diagram)

eg:

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	

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

$= ab + c$

Maxterm Representation:

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

Shannon's expansion

$$f(x_1, \ldots x_n) = x_i \cdot f(x_1, x_2, \ldots, 1, \ldots n) + \overline{x_i} f(x_1, \ldots 0, \ldots x_n)$$

co-factor → f_{x_i} ; co-factor → $f_{\overline{x_i}}$

⇒ if $z = ab + c = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$

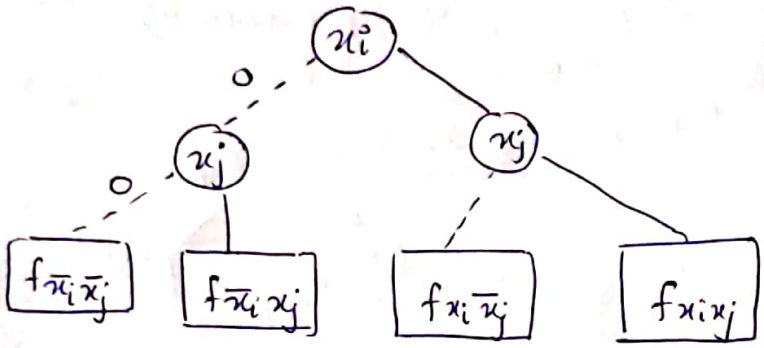
$f_a = b + c$

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

Further,

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

Graphical Rep:

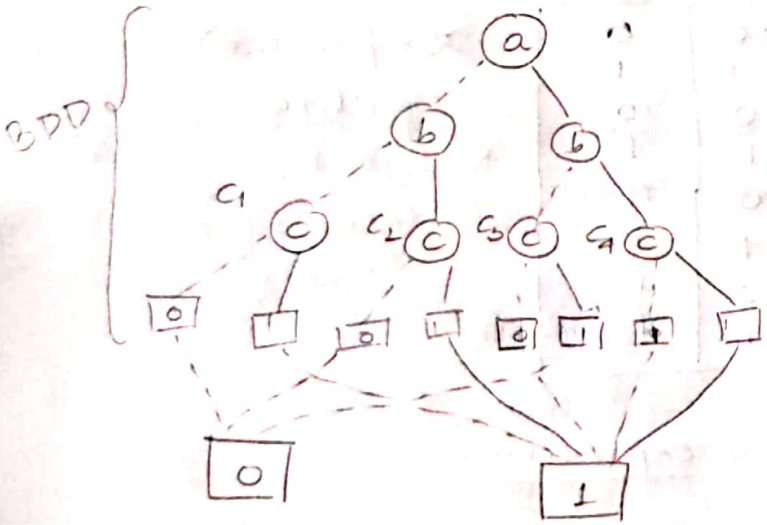


Now, $x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i} = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$

since ↓ & ↓ are always complement

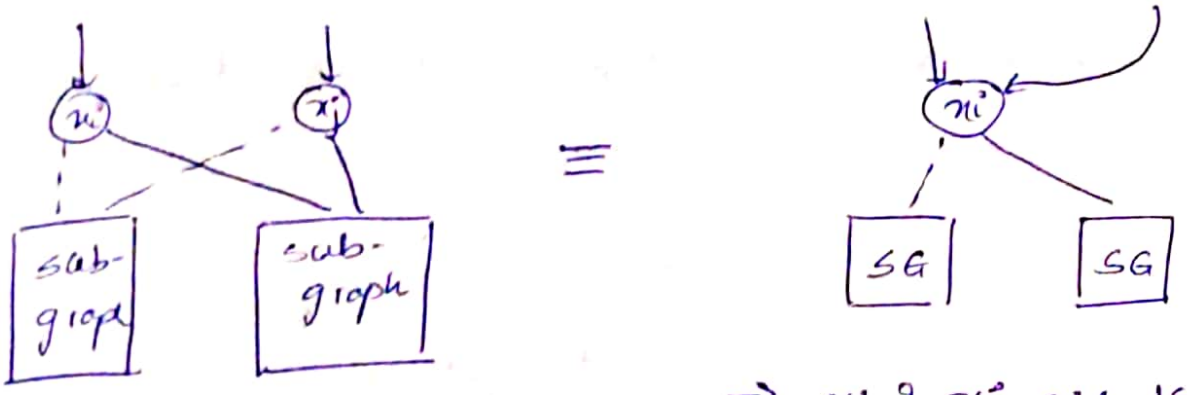
so we can build a 'Binary Decision Tree'

⇒ $2^{n+1} - 1$ space — sum of geometric progression



(we know redundancy present here)

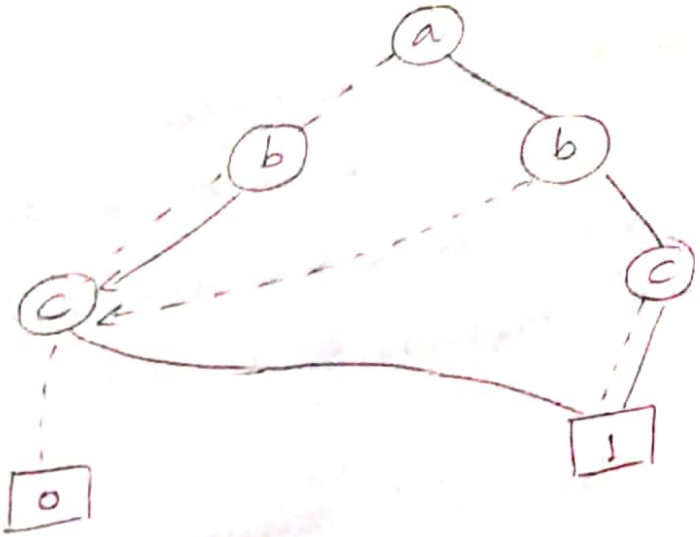
⇒ since output on zero or 1 terminal occurs ~~at~~ only 2 outputs

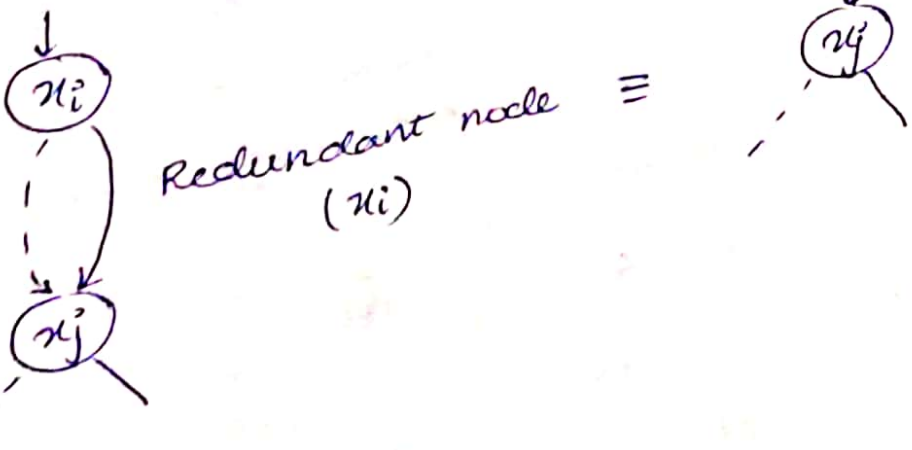


⇒ x_i & x_j are ISOMORPHIC NODES

⇒ G_1 & G_2 are isomorphic

⇒





⇒

• if $a > b > c$ ⇒ 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 fixed

- 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 $\boxed{0} \equiv f^{OFF}$, $\boxed{1} \equiv f^{ON}$, thus it contains complete info of truth table
- in ON or OFF any variable can occurs once or not at all. Also they occur in the same order

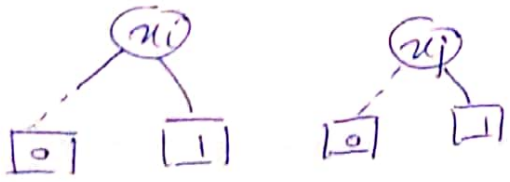
→ For the reduced diagram

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

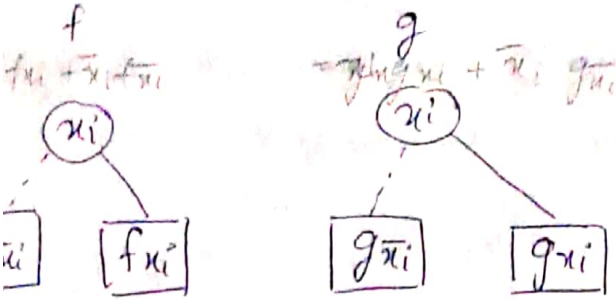
$$f^{OFF} = a\bar{b} + \bar{a}\bar{c}$$

→ BDD requires double space than truth table

→ so we need to do an incremental build up

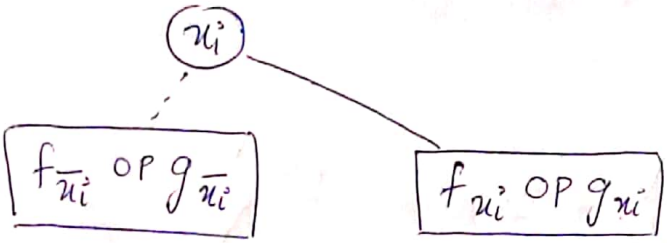


⇒ using these we should be able to generate $x_i x_j$



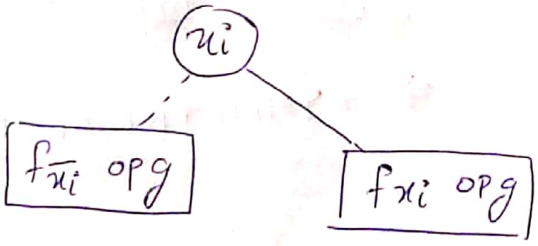
$$f \cdot g = x_i f_{x_i} g_{x_i} + \bar{x}_i f_{\bar{x}_i} g_{\bar{x}_i}$$

$$f \text{ op } g = x_i (f_{x_i} \text{ op } g_{x_i}) + \bar{x}_i (f_{\bar{x}_i} \text{ op } g_{\bar{x}_i})$$

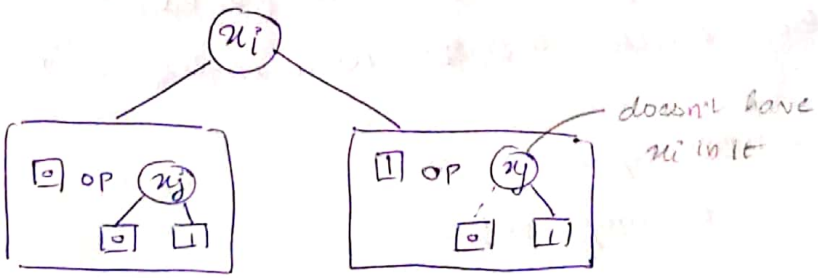


[do this recursively]

if g doesn't have x_i:

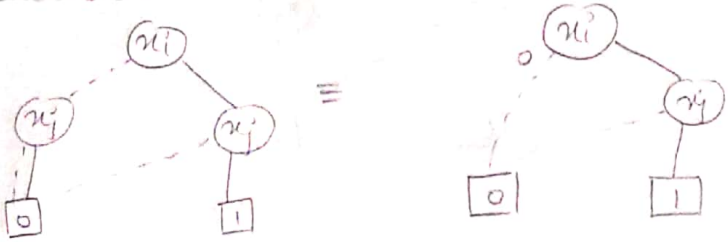


⇒ if $x_i > x_j$



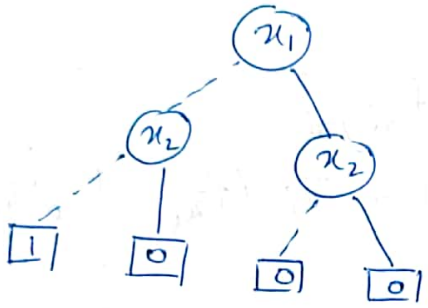
if op = AND

then apply reduction:

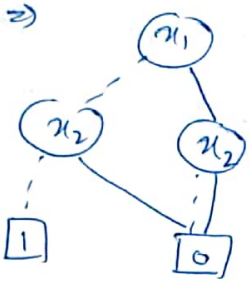


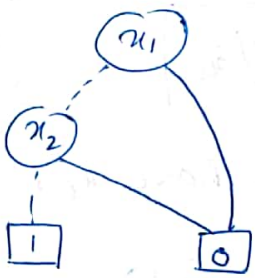
- if now we want $\overline{x_i \cdot x_j}$ we just interchange 0 & 1, since outputs now complemented

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



$\Rightarrow \bar{x}_1 \cdot \bar{x}_2$





$\Rightarrow$ SYMMETRICAL F^N

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

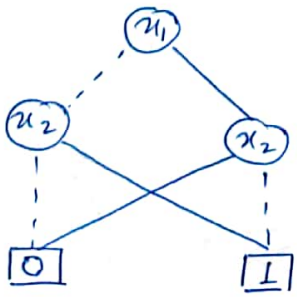
adds only 19 extra nodes for BDD → each variable adds 2 extra nodes

But truth table has 1024 nodes

01/10

- BDD represent ckt
  i) compactly

eg: $x_1 \oplus x_2$



BDD : after every operation apply reduct^n

- grows exponentially for multiplier kind of ckt

→ $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \ldots$: 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

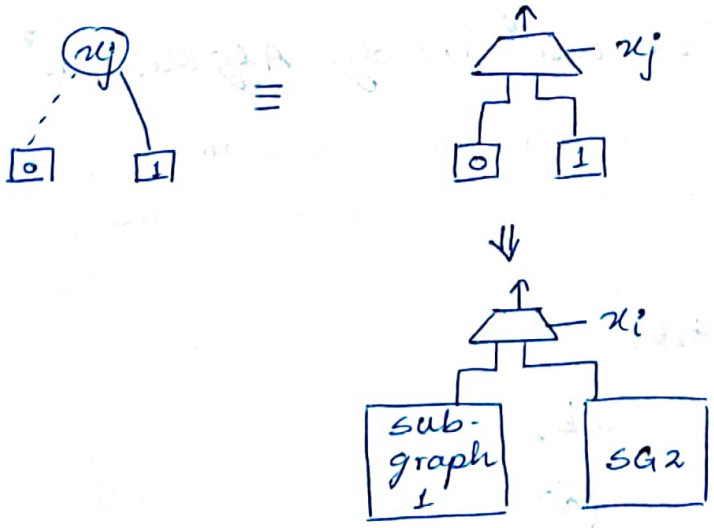
eg: carry look ahead adder & ripple carry adder should give the same BDD)

Ripple carry adder easier to verify, so it acts as a template to check correctness of other adders

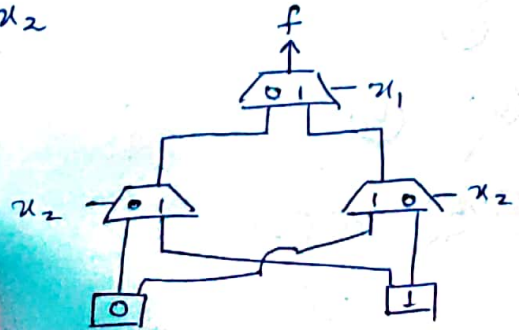
→ fully autonomous, human intervention not needed

→ compare BDD generated from boolean f" & gate level netlist

- **Synthesis** : directly generate ckt from BDD



→ $x_1 \oplus x_2$



" MULTI LEVEL LOGIC "

MUX is a universal

- Size of ckt depends on
  - no. of nodes (= no. of MUX's)
- Performance
  Max
  ∴ Delay of longest path = t_m × no. of variable
  (t_m → MUX delay)

⇒ we want to
- minimize no. of nodes (area reduction)
- minimize depth of ckt. (delay reduction)

- No. of nodes depends on order of variable.

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

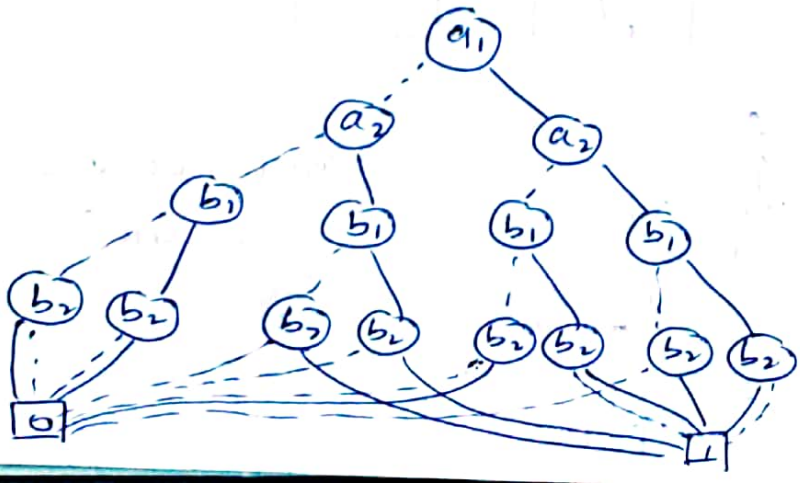
  Exploring all order = n!

  → Hence we use "Greedy Algorithm"

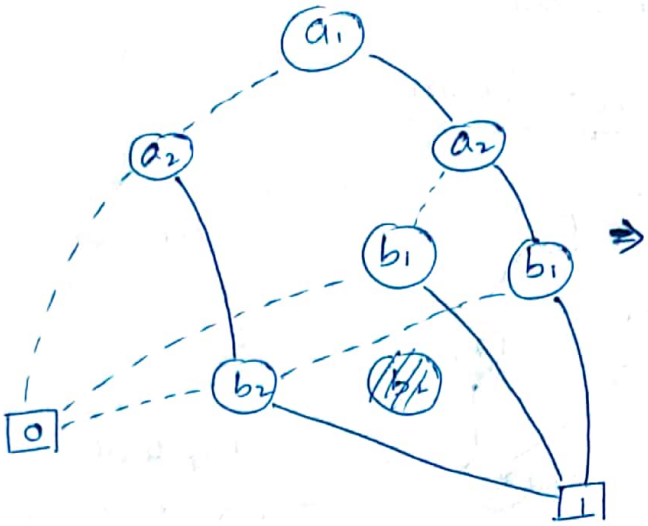
  [If not optimum, extra MUX's use more area and power → becomes a recurring cost for

eg: $f = a_1b_1 + a_2b_2$

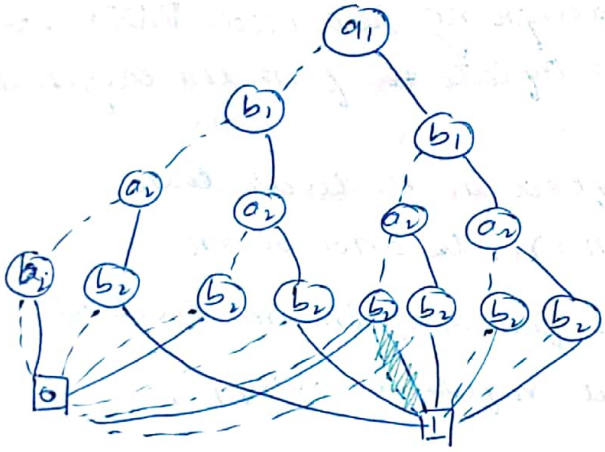
a) $a_1 > a_2 > b_1 > b_2$

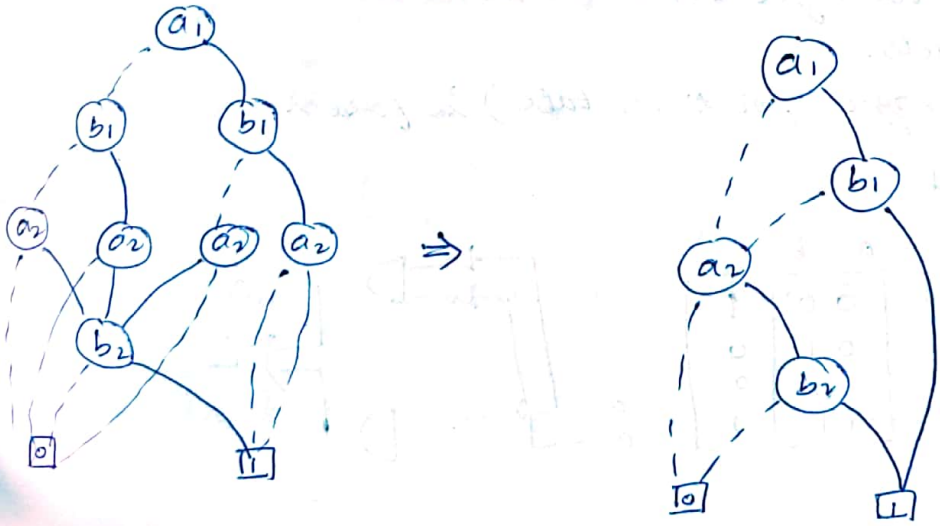


a)



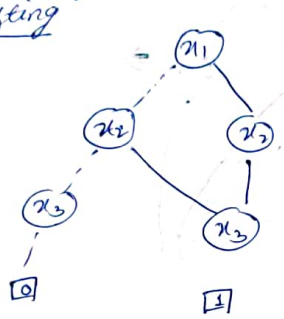
b) $a_1 > b_1 > a_2 > b_2$ — can make decision in 1st 2 levels





03/10

- To choose optimal order
  - ~~swapping~~ sifting algorithm



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

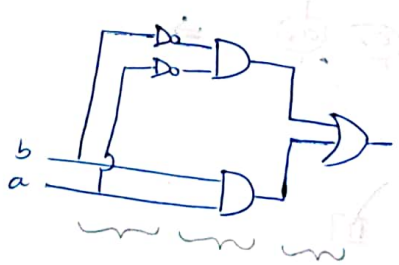
- Since we keep accepting reduced BDD, so no way we can inflate ~~the~~ from the original ~~order~~ BDD
- 1st variable sifted in n levels, then next sifted in n-1 levels and so on
- once variable 1 finds optimum level we don't change it in further swapping

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

2-level logic (for truth table) is faster

eg: $a \overline{\oplus} b$

	a	b	
Implicant	0	0	1
	0	1	0
	1	0	0
Implicant	1	1	1



Variable

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

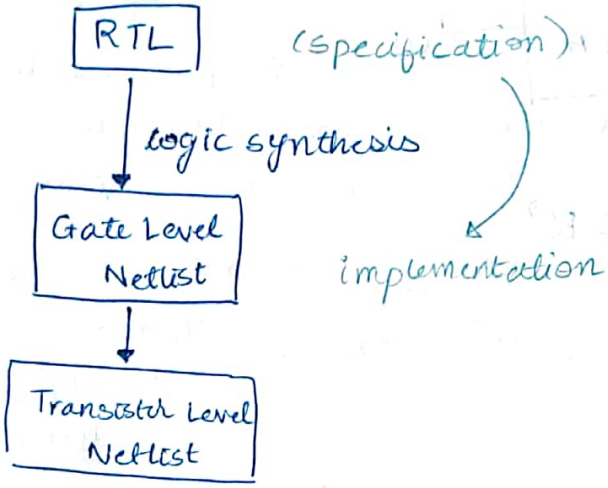
2 levels if all literals (complements) not present

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
  ↓
  ROBDD
  ↓ Replace Node by MUX
  Gate level netlist

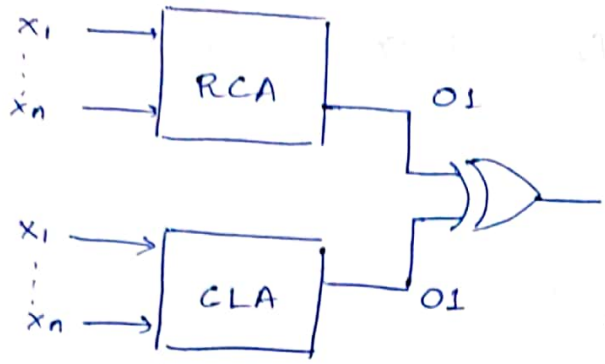
  Do we need to check for equivalence? (ROBDD mathematically correct)

- Truth Table
  ↓
  minterm
  ↓ K-Map
  ckt

  should be equal since K-Map reduction is mathematically correct

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

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



'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 don't equal
(this input called COUNTER EXAMPLE)

- Satisfiability problem ↘ transforms to

Now, Boolean expression can be written as POS or SOP

eg. ↘ clause

$$\underbrace{(a+b+c)}_{C_1}\underbrace{(\bar{a}+\bar{b}+\bar{c})}_{C_2}\underbrace{(\bar{a}+\bar{c})}_{C_3}=1$$

check if satisfiable
- computationally harder to detect

to check equal to 1 POS is better coz all C_1, C_2, C_3 have to be = 1

→ if we find atleast one set of inputs such that it equalicates 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 ; 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 $c_1 = 0$

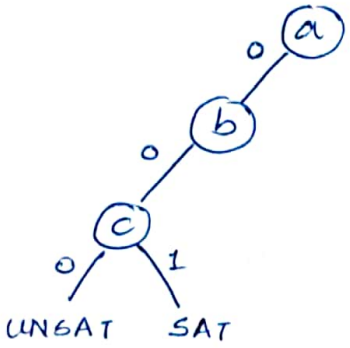
⇒ satisfied

⇒ 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 solvers

→ Need to express f in to POS

→ To check equivalence of 2 ckts (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: $a \rightarrow \bar{b}$

  $\bar{a} \rightarrow b$

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

→ 

Behaviour: $((a=0) \rightarrow (c=0))$ $(\bar{a} \rightarrow \bar{c})$

$((b=0) \rightarrow (c=0))$ $(\bar{b} \rightarrow \bar{c})$

$((a=1) \cdot (b=1) \rightarrow (c=1))$ $(a.b \rightarrow c)$

{ Implication Relationship }

$X \rightarrow Y \equiv \bar{X} + Y$

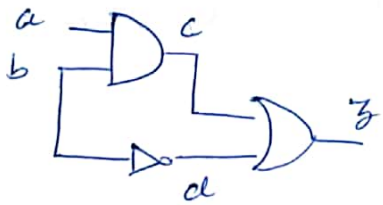
⇒ AND $\equiv (a+\bar{c})(b+\bar{c})(\bar{a}.\bar{b}+c)$

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

Inverter = 2 clauses

AND = 3 —"—

XOR = 4 —"—



$((a=1) \rightarrow (c=1))$ $(a \rightarrow c)$

$((b=1) \rightarrow (c=1))$ $(b \rightarrow c)$

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

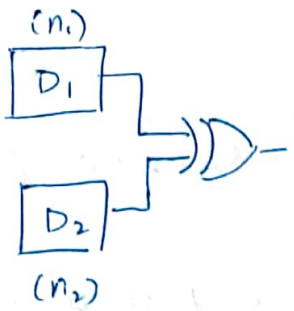
⇒ $(\bar{a}+c)(\bar{b}+c)(a+b+\bar{c})$

⇒ Therefore for above ckt,

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

$(\bar{c}+z)(\bar{d}+z)(c+d+\bar{z}) \cdot z$ → if we want output = 1

⇒ if we have n no. of gates then roughly we get 3n no. of clauses



⇒ No. of clauses =

$3n_1 + 3n_2 + 4$

#10 SAT

( )( ) ( ) = 1

c1 c2 c3

Literal : variable in complemented or uncomplemented form

- DPLL (1960)

↓

assgn 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

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

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

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

, here b is also in pure form

⇒ $a = 1, b = 0$

② Unit clause rule (1995)

$(\bar{a}+b+c)^{C1}\ (\bar{a}+\bar{b}+e)^{C2}\ (b+\bar{c}+d)^{C3}\ (\bar{a}+e)^{C4}\ (\bar{e}+d)^{C5}$

Since a appears in C1, we put $a=1$

⇒ C4 turns into clause with one literal

⇒ $e=1$ — unit clause

⇒ C5 turns into a unit clause

⇒ $d=1$

→ propagate these unit clauses, called "Boolean constraint Propagation" (BCP)

→ so rather than assigning in chronological order, we use BCP

③ Consensus theorem

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

→ either one of b or c should be 1

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

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

1 1 — Pure literal

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

④ Stallmarc's Method [satisfying literal by literal]

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

C_1 C_2 C_3 C_4.

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 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 : $d=1$

$\Rightarrow$ $d=1$ = "Essential Assignment"

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

⑤ Recursive learning (RL)

- does similar job but clause by clause

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

for satisfying C1

→ a=1 ⇒ c=1 ⇒ d=1
→ b=1 ⇒ d=1

} ∩ = d=1 → essential assgn

for satisfying C2

→ a=0 ⇒ b=1 ⇒ d=1
→ c=1 ⇒ d=1

} ∩ = d=1

⇒ Build tree for all variables that are not essential assignment

⑥ Local Search Method

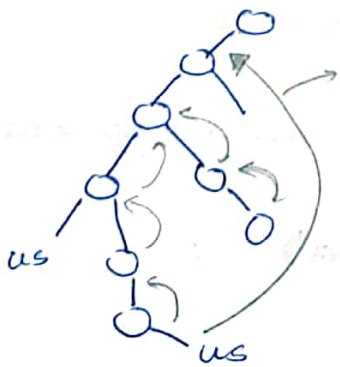
- Suppose we do a random assgn

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

  ⇒ C4 is UNSAT → C3 also

  ⇒ Need to flip some variable

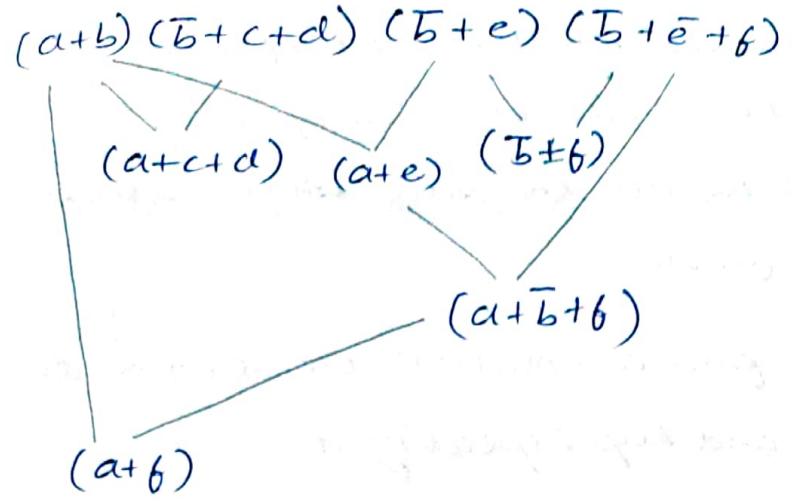
  ⇒ we flip that variable which appears in more no. of clauses

- Find essential ~~assgnts~~ variable and assgn them high priority then explore the tree
- Simple approach would be to find the variable which appears in most clauses and prioritize it
- 

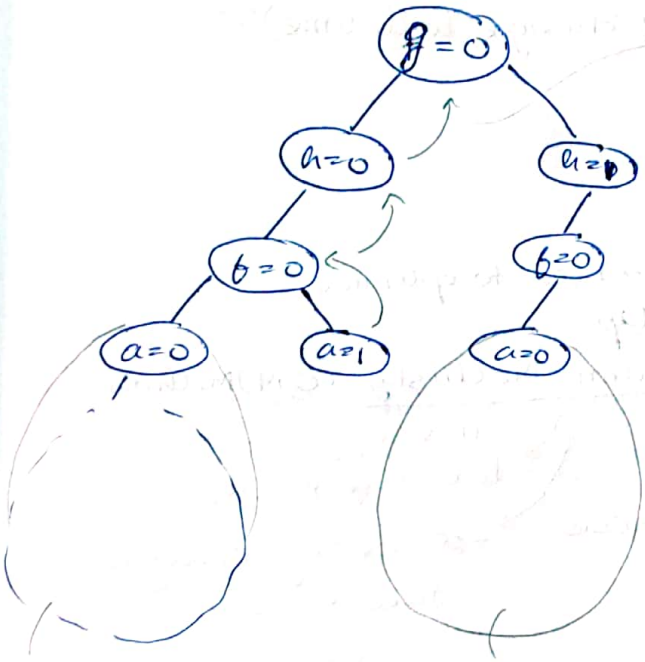
some way to get to the root problem rather than going chronologically

↑ Non-chronological Backtracking

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



→ consensus helps us make additional clause



→ suppose those values don't satisfy

this entire subtree doesn't pass

don't want to explore this entire subtree again

⇒ equivalent to adding additional clause $(a+f)$

⇒ '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

$f_1 \equiv f_2$

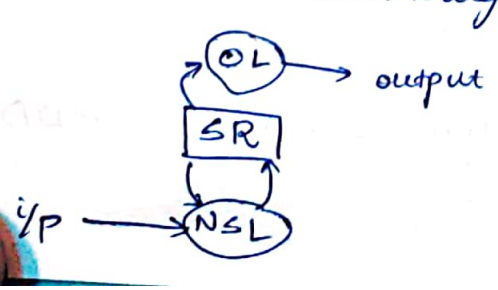
static f^n (doesn't change with time)

Designing

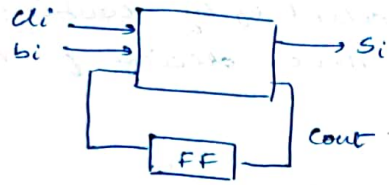
→ Truth table
→ Minterm → need to optimize, K-Map
→ Maxterm → need to optimize, K-Map
→ ROBDD → Synthesis

Quinn-McClusky (QM) Method
(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

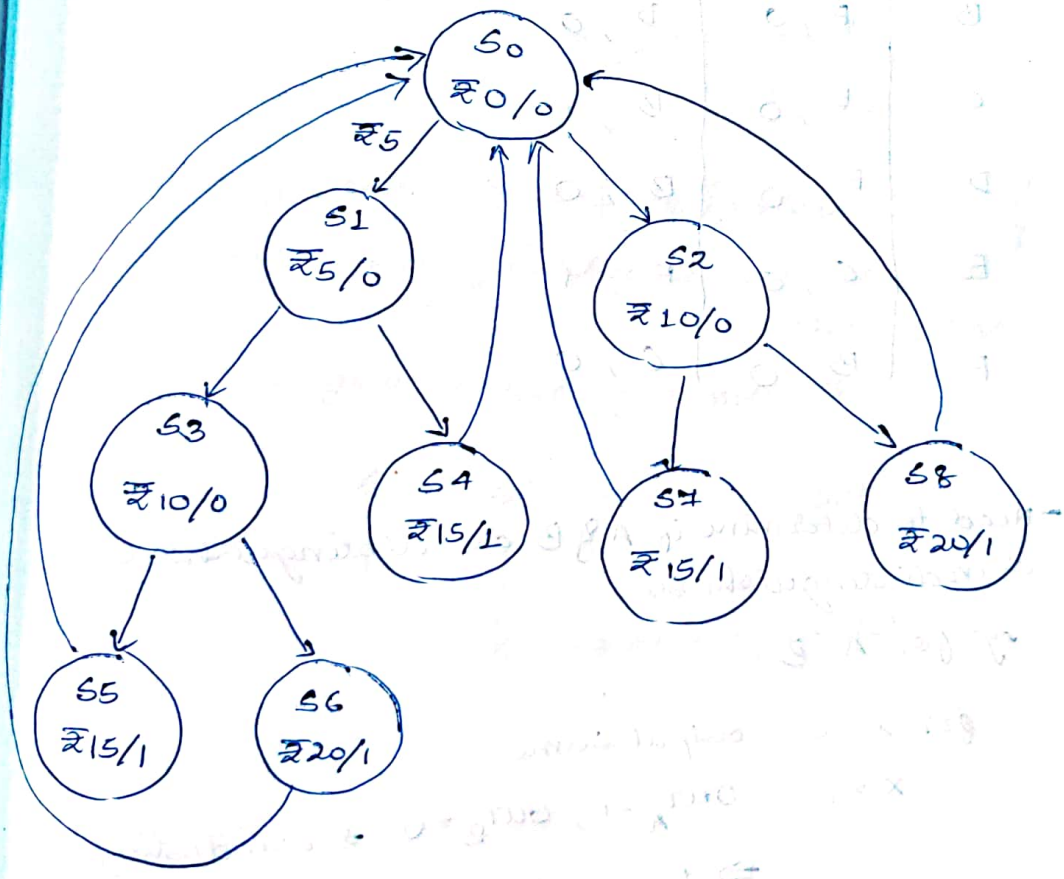


- Adder:
  - can replicate a fulladder in series (like RCA)
  - or make a sequential adder



⇒ output depends on history

eg: we have a coffee dispenser machine



Hence finally we want to dispense coffee, state S5, S6, S4, S7, S8 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	Next state, 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

eg: for A, B

for X = 0 output same

X = 1 $OUT_A = 1$, $OUT_B = 0$ ⇒ can distinguish

⇒ 1 distinguishable

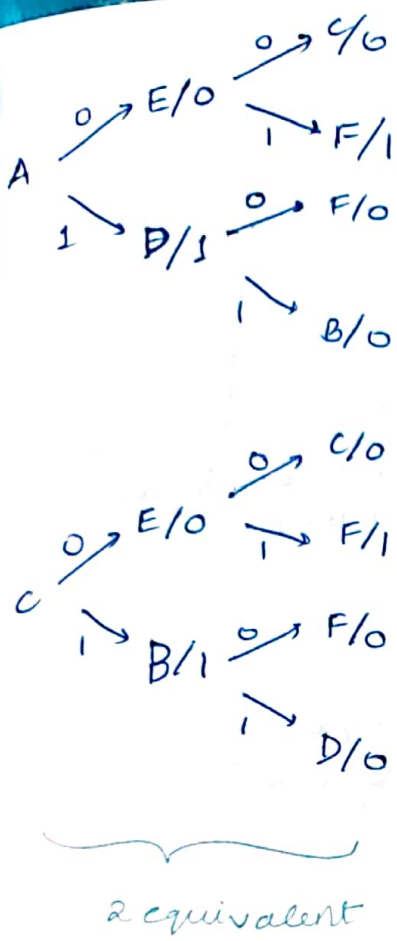
⇒ 0 equivalent

A & C

X = 0 same

X = 1 same } not 1 distin.

⇒ 1 indistin



- K distinguishable ⇒ K-1 equivalent
- 9 never indistinguishable ⇒ always equivalent

→ But go on indefinitely to notice them

$S_i \equiv S_j$ → K equivalent

S_n S_m → also equivalent

(⇒ next state are equivalent)

⇒ S_i & S_j are always equivalent.

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

- **Partitioning**

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

$P_1 = (ACE)(BDF)$

(ACE) (BDF) — indistinguishable by applying 1 input

$P_2 = (ACE)(BD)(F)$ — their next state lies in the same partition

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

$P_4 = (AC)(E)(BD)(F)$ — keeps repeating so stop here

Taking the example of P_2

	NS(0)	NS(1)	
A	E	D	same partition (ACE)
C	E	B	same partition (BDF)
E	C	F	
B	F	D	
D	F	B	
F	B	C	

→ We run this until
- we get the sequence to be equal
- number of states present

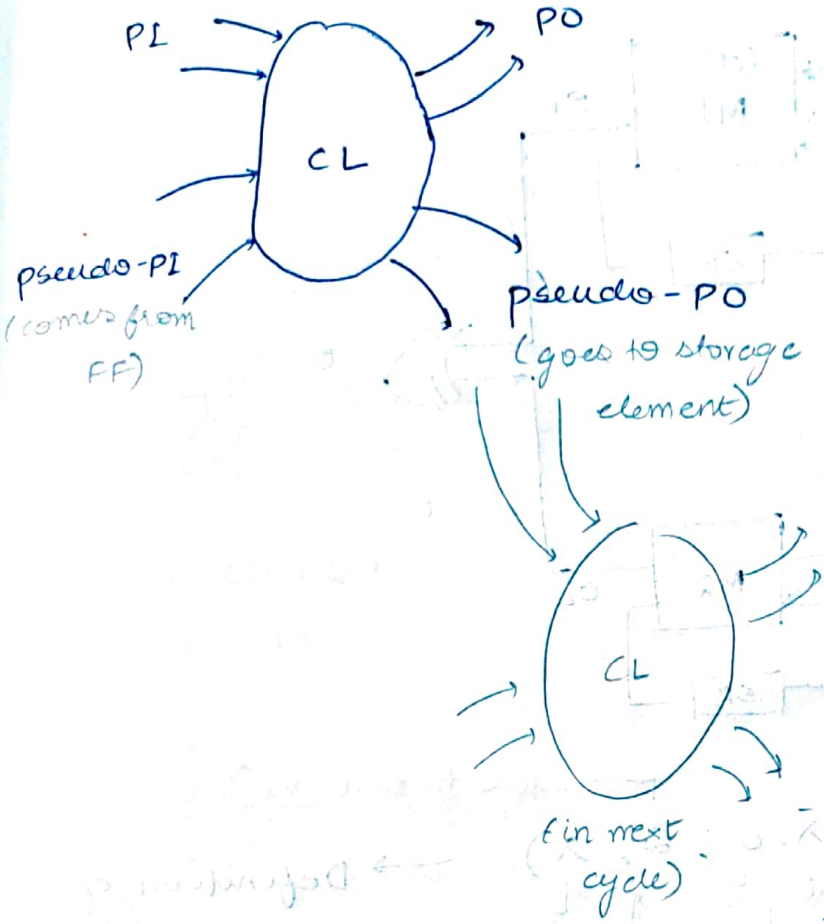
- checking equivalence of 2 state machine

eg: 2 machines

M_1 ↓ (Reduced machine) M_1'

M_2 ↓ minimize machine M_2' ⇒ Isomorphic (check)

Isomorphic : if 2 graphs are same



.... can keep unrolling
⇒ all the resources keep replicating
⇒ complexity grows

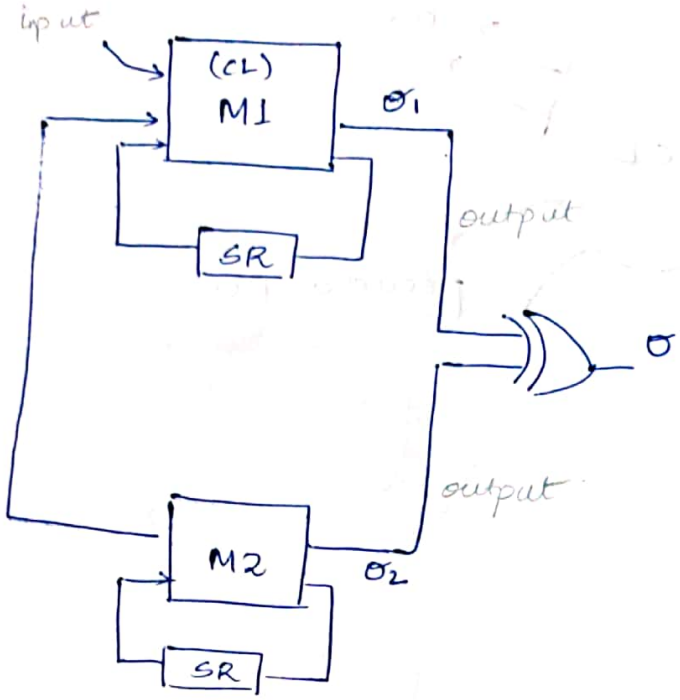
We start from a known state
→ reset state

{ Unrolling ≡ Time Frame Expansion }

If we have m outputs and we unroll n no. of times
⇒ m×n outputs
⇒ which need to be XOR'd by the Miter

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

⇒ 'Sequential Equivalence Checking'



M (S, S_0, X, O, δ, λ) → Definition of a machine

- S — State Variable (set of states)
- S_0 — initial state
- X — inputs
- O — output variable
- δ — state transition fn
- λ — output fn

M1: $(S', S_0', X, O, \delta', \lambda')$

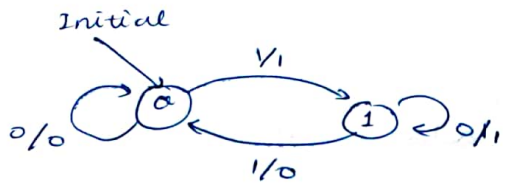
M2: $(S^2, S_0^2, X, O, \delta^2, \lambda^2)$

→ Total number of states = $|S'| \times |S^2|$ — hence called 'PRODUCT MACHINE'

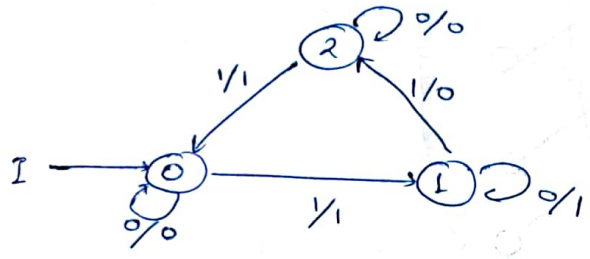
$$O = \begin{cases} 0 & \text{when } O_1 = O_2 \\ 1 & \text{o.w.} \end{cases}$$

→ we do a reachability analysis
↳ whether all states are reachable or not

Eg:



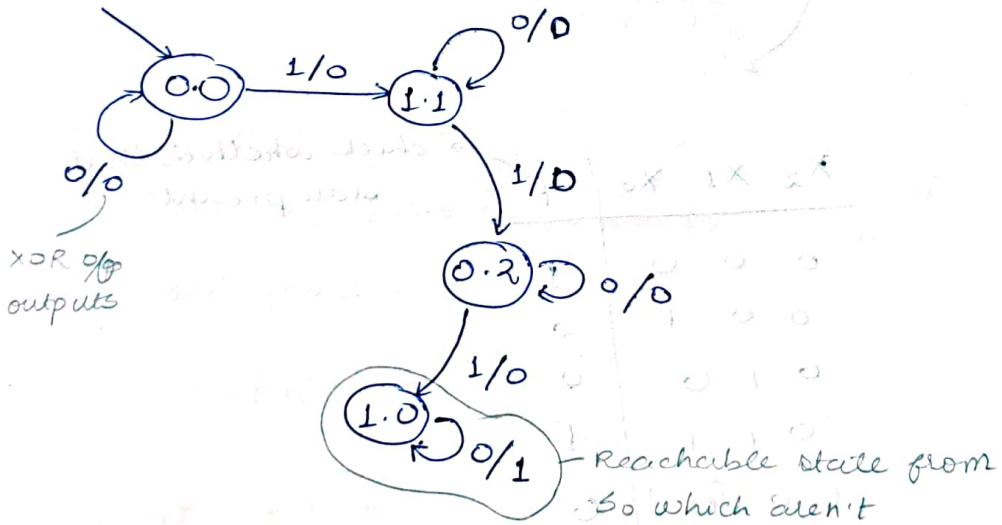
: M1



: M2

Total states = 6

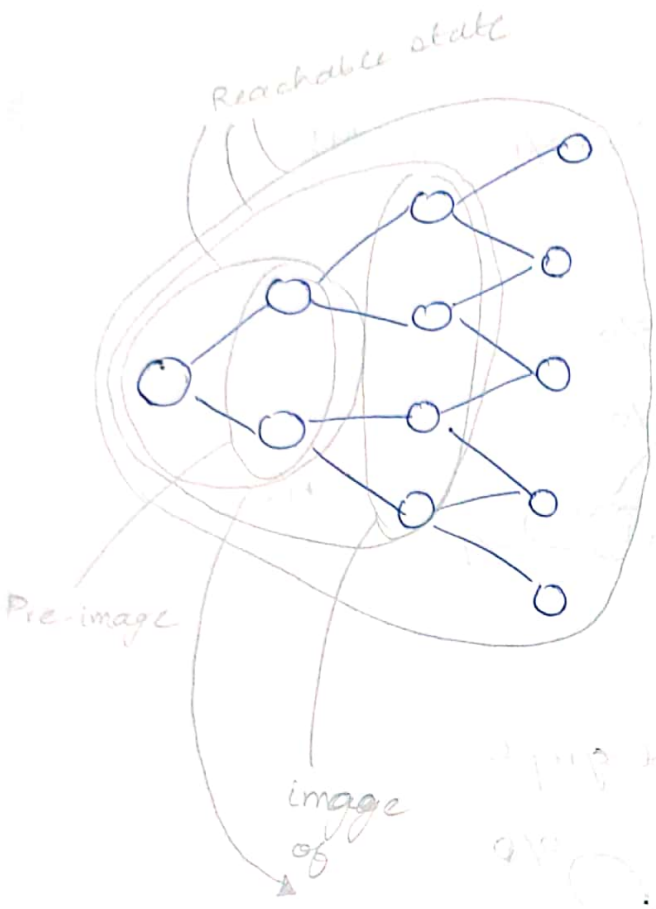
→ Making the combined graph



⇒ 1 1 1 0 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
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

→ check whether that state present

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

transition to next state

x	R_1 b_0	t_1 t_0	F
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

F → checking which transition is valid

→ set of reachable states

Transition Relationship: $R(x, s, t)$

$C(s)$: set of states from where we can start

$T(x, s, t) = R(x, s, t) \cdot C(s)$ → set of reachable states from $C(s)$

- Eliminating a variable

$$f(x_1, x_2, \ldots x_i, \ldots x_n) = x_i f_{x_i} + \overline{x_i} f_{\overline{x_i}}$$

if we eliminate x_i

$$f = f_{x_i} + f_{\overline{x_i}}$$

eliminate x & then s and get $T(t)$ (image)

$$\exists_{x_i} f(x_1, \ldots x_n) = f_{x_i} + f_{\overline{x_i}}$$

Existentially quantify

$$T(x, s, t) = \exists_x \exists_s T(t)$$

then is 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 equivalent

10/10 Equivalence checking of 2 state machines

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

Reachability Analysis (in a Product machine)

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

$$\theta = \begin{cases} 0 & , (\theta_1 = \theta_2) \\ 1 & , \text{o.w.} \end{cases}$$

- can store as a boolean function
  - ↳ store as ROBDD

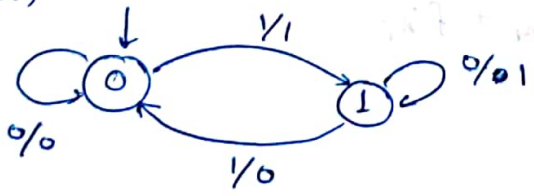
$C(k)$ = set of all reachable states by that time

Transition Function:

$R(X, s, t)$

↳ represent as ROBDD

eg; (a)



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

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

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

S_0^1 = initial state

= $\overline{s_0}$

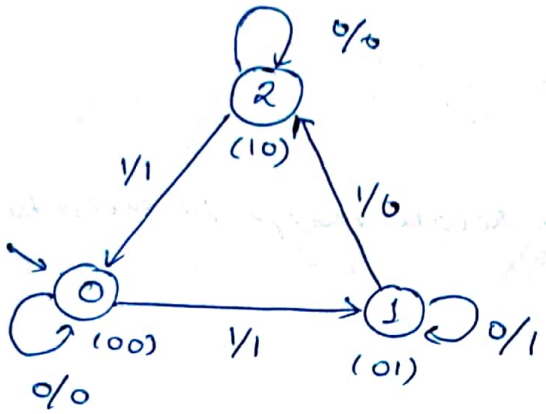
x \ b_0	0	1
0	0	1
1	1	0

$\Rightarrow S'(x, b) = \bar{x} b_0 + x \bar{b}_0$ (next state)

x \ b_0	0	1
0	0	1
1	1	0

$\Rightarrow \lambda(x, b) = \bar{x} b_0 + x \bar{b}_0$ (output)

(b)



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

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

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

$b_2 b_1$ \ x	0	1
00	0	1
01	1	0
11	x	x
10	0	0

writing the LSB of state

$\Rightarrow S_1^2 = \bar{b}_2 \bar{b}_1 x + b_1 \bar{x}$

$b_2 b_1$ \ x	0	1
00	0	0
01	0	1
11	x	x
10	1	0

next state MSB

$\Rightarrow S_2^2 = b_2 \bar{x} + b_1 x$

$s_2 s_1$ \ x	0	1
00	0	1
01	1	0
11	x	x
10	0	1

$$\lambda^2 = s_1\bar{x} + \bar{s}_1 x$$

$R(x, s, t) \Rightarrow$

$s = s_2 s_1 s_0$

$t = t_2 t_1 t_0$

our transition relationship should be such that:

$t_1 \equiv S_1^1$

$t_2 \equiv S_1^2$

$t_3 \equiv S_2^2$

$$\Rightarrow R(x, s, t) = (t_1 \equiv S_1^1)\cdot(t_2 \equiv S_1^2)\cdot(t_3 \equiv S_2^2)$$

$$= \prod_i (t_i = S_i^*)$$

[Equivalence is XNOR operation

$a \equiv b \Rightarrow a \odot b = ab + \bar{a}\bar{b}$]

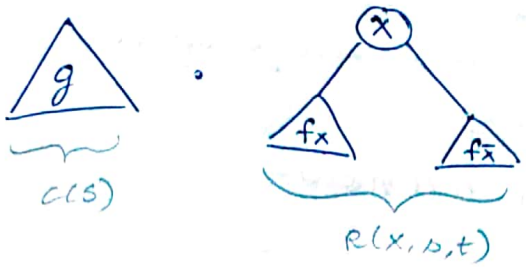
$$= (t_1 \equiv s_0\bar{x} + \bar{s}_0 x)\,(t_2 \equiv \bar{s}_2\bar{s}_1 x + s_1\bar{x})\,(t_3 \equiv s_1 x + s_2\bar{x})$$

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

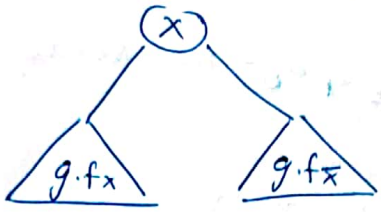
$$T(x, s, t) = R(x, s, t)\cdot C(s)$$

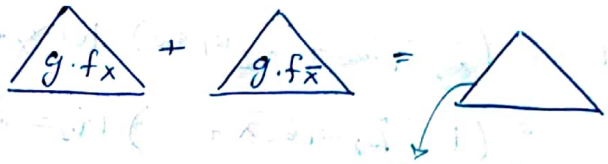
[We know the BDD of individual paranthesis and hence can combine and reduce]

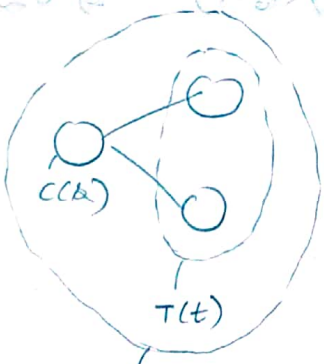
want to existentially quantify x



=



$\Rightarrow \exists_x T(x, s, t) =$  further eliminate $s_2 s_1 s_0$ to get fn of $t_2 t_1 t_0 = T(t)$



→ to get next set of reachable states

$$T = (\bar{s}_2 \bar{s}_1 \bar{s}_0)(t_1 \equiv s_0 \bar{x} + \bar{s}_0 x)(t_2 \equiv \bar{s}_2 \bar{s}_1 x + s_1 \bar{x})$$
$$(t_3 \equiv s_1 x + 0 \cdot \bar{x})$$

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

at $x = 0$; $T = (t_1 = 0)(t_2 = 0)(t_3 = 0) \Rightarrow 00$

$x = 1$; $T = (t_1 = 1)(t_2 = 1)(t_3 = 0) \Rightarrow 11$

because currently we are in $\bar{s}_2 \bar{s}_1 \bar{s}_0$ 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. $\bar{s}_2 \bar{s}_1 \bar{s}_0 + \bar{s}_2 s_1 s_0$ = new state

($\bar{s}_2 \bar{s}_1 \bar{s}_0$: generates the same states; $\bar{s}_2 s_1 s_0$: generates new states)

$$T = (\bar{s}_2 \bar{s}_1 \bar{s}_0 + \bar{s}_2 s_1 s_0)\, R(x, s, t)$$

$$= (t_1 \equiv \bar{s}_2 s_1 s_0 \bar{x} + \ldots)(t_2 \equiv \bar{s}_2 s_1 s_0 \bar{x} + \ldots)$$

$$(t_3 \equiv \bar{s}_2 s_1 s_0 x)$$

at $x = 0$: $T = (t_1 = 1)(t_2 = 1)(t_3 = 1)$ ⇒ 11

$x = 1$: $T = (t_1 = 0)(t_2 = 0)(t_3 = 1)$ ⇒ 02