22/1/20

Janak Patel

- 'Manufacturing Test' for a chip
  - ⟶ Only Pass/Fail : No diagnosis
- Defects : during fabrication

- Testing hierarchy:
  - High Level Program
  - ↓
  - Machine Level / ISA (cycle-accurate)
  - ↓
  - RTL Level (clock-accurate)
  - ↓
  - ✓ Logic Level, Gate Level
  - ↓
  - Circuit Level : Transistors, wires
  - ↓
  - Physical Level : Silicon Layout.

(Defect)

- Defects —produce→ computation error
  - Detect error ~~on~~ using one of these levels.

- High level tests do not provide enough coverage ∴

* Functional Test:
  eg: 'Does addition work?'

nMOS conducts with V_G = high ($V_{GS} > V_T$)

low V_{GD}

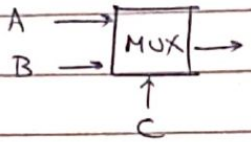
P

→ Logic Stuck at 0 and 1

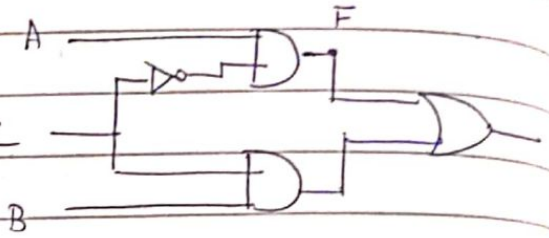
- Wide coverage ☺

eg MUX

RTL Level



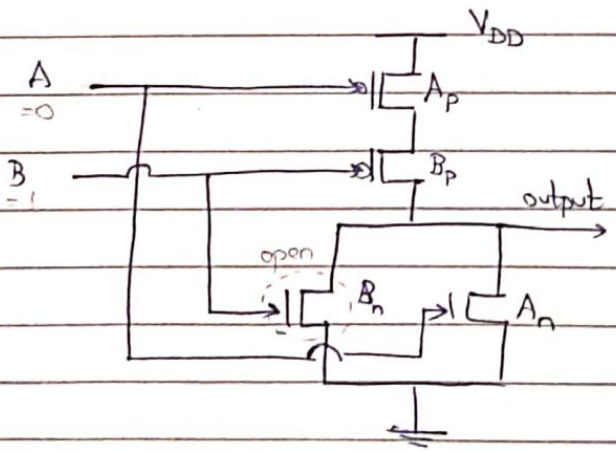
Logic level



- Inputs and outputs of all gates are possible fault sites.

- To test stuck-at-x, apply $\bar{x}$.

→ Transistor stuck-open or stuck-close



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

passive

In this case, the output remains floating ("high impedance state")

Z

If there is some capacitor at the output, it will retain its previous state

To test, try to get B_n to conduct with appropriate inputs.

A = 0, B = 1

Case 2 B_n is shorted. (closed)

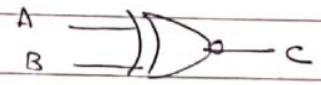
To test, try to make it open. (A=0, B=0)

Output is 'indeterministic' (N)

↳ active, driven

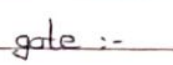
* These faults may or may not produce a final logical error.

→ XNOR gate :



To test all faults, we would think that the 3 vectors A, B, C are sufficient to produce necessary combinations.

However, XNOR is not a basic gate. When expressed in terms of constituent AND, OR, NOT, we would require 4 vectors to test all faults.

eg AND gate :- 

Total 6 possible logic faults :- A, B, C stuck at 0, 1

'Fault Dictionary'

A	B	Correct	A/0	A/1	B/0	A_p open	A_p closed .....
0	0	0	0			Z	
0	1	0	0			Z	
1	0	0	0			Z	
1	1	1	0			Z	

256

Every output can be one of {0, 1, N, Z}. There are 4 outputs here.

256 combinations. (255 are defective)

XNOR

If only 1st outcome is defective: 1, 0, 0, 1 instead of 0, 0
we would need to test with A=0, B=0.

AND

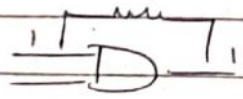
Axiom: No defect can transform a simple gate into a higher order gate
↳ XNOR.

Hence, we never require A=0, B=0 case for testing AND

→ Interconnect Faults.

1. Opens
2. Shorts

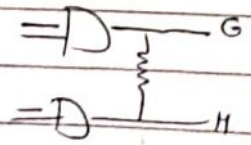
Input to output short



Input to input short

Resistive short or zero-resistance short

Output to output short



Whichever gate is more powerful will drive the outputs

↗ Fault is undetected when H = 0

Test G/1 :- Make G=0. H=0 or 1 with probability 0.5

∴ Can detect short with 50% probability

50%

Hence, after testing G/0 and G/1, short will be detected up 75% and not detected up 25%

Coverage = 75%

Now, also test H similarly.

Coverage → 100% (~99%) with multiple testing trials

Hence, 'stuck-at' fault model is sufficient to tackle interconnect faults

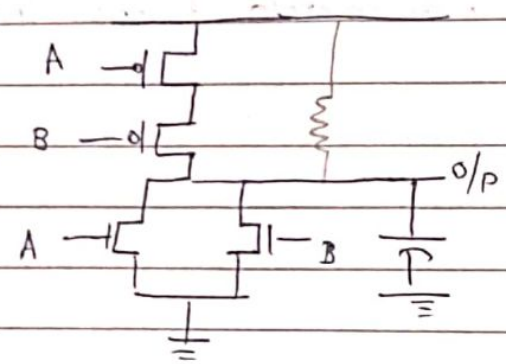
24/1

DELAY FAULTS

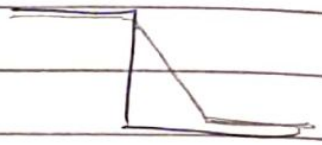
I DEFECTS

- Can cause timing violations without changing logic function

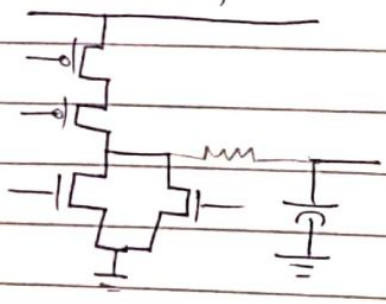
eg - Resistive Bridges

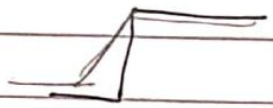


Switching



eg - Resistive Opens





eg - Capacitive Coupling

?

- Miscalculation of functional power :-
IR drop in power lines during simultaneous switching of many gates lead to slower transitions.

A] Process Variations

1. Line Spacing → Coupling faults
2. Line Thickness → Increased delay
3. Gate threshold variations → Increased transition time

B] Aging

1. Metal migration → ↑ Resistance, Delay
2. Gate oxide degradation → ↑ Gate threshold, Transition time
3. NBTI / PBTI → V_{TH} degrades, but can recover
   - NBTI: Negative bias temperature instability (pMOS)
   - PBTI: Positive (nMOS)

II DELAY FAULT

≜ There exists a logical path from input to output on which propagation time of transition exceeds specified time period.

- Single gate may be out-of-spec. No fault if total delay is under specified period.

- Hard shorts and opens : Logic stuck-at testing ✓✓

Resistive shorts/opens, coupling, aging, process variations : Delay testing ✓

→ Delay Fault Testing:
- All path delays should be less than clock period.

→ Scanning
- Test vector is applied via shift registers
  'Scan chain'
- Response of all flip flops is scanned as output.

→ Two Vectors Delay Test

?

→ Delay Fault Models
↓
Logic delay, not circuit delay.

1. Transition faults. Large delay at one node
   Any transition through this node will be delayed past clock period.

2 Path Faults: Distributed delay, from latch to latch

- Transition fault: Test by stuck-at model at that node.
  ① Initialize that node to 0 or 1
  ② Test that node for stuck-at 1 or 0.
  "Two pattern verification"

☺ May detect delay defects missed by stuck-at tests
☹ ✗ Distributed delays, small delays (→ majority.)
✗ Process variations

- Path delay fault:
☺ More likely to detect small delays
☹ Complex, low coverage.

Try to give a transition at input that can generate transition at output
- May not always be possible.

→ Path Tests
1 Robust - Guaranteed delay fault detection on targeted path independent of other delays.

2 Non-robust - Guaranteed " " " " "
if no other delay is increased.

- Requires weaker test conditions

eg Robust Test Conditions

0-to-1 transition 1-to-0 transition

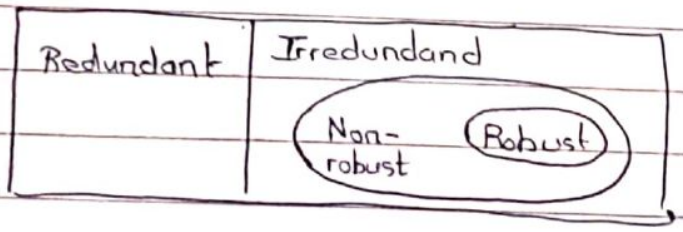
eg Nonrobust Test Conditions

→ Functionally Sensitizable Path

≜ AND gate with 0-to-1 transition, with side inputs '1'
or OR 1-to-0 0

- Functionally un-sensitizable paths have no effect on timing and are redundant.
They cannot be robustly tested

- Also, very small portion of irredundant paths are robustly testable



→ ~~Functional Transition~~ Method

- Initialization vector, followed by transition vector.

1. Functional Transition Method:
Transition vector is functionally generated from initialization

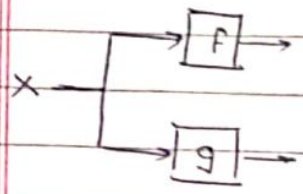
2. Launch-on-shift TM:
Transition vector is initialization vector shifted by one bit
- Restrictive in

→ Segment delay fault model
↳ section of a path
- Consider transitions on segments of length L (parameter)

→ Critical Path Testing

29/21

I EQUIVALENCE CHECKING



$\forall x$, is $f(x) = g(x)$?

1. Truth Table (Brute Force) : Not scalable.

2. Standardize in sop/pos form (canonical representations)
Find minterms

* $F: B_2^n \longrightarrow B_2$

inputs
No. of ~~atoms~~ in $B_2^n = 2^n$
No. of possible functions = $2^{(2^n)}$

Still not scalable.

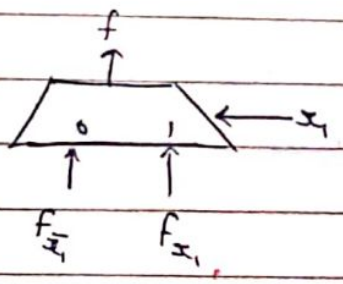
3. ROBDD : Reduced Order Binary Decision Diagram.

* Shannon's Expansion:

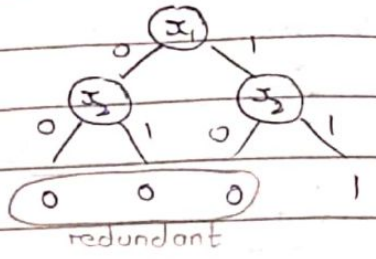
$$f(x_1 \dots x_n) = x_1 \cdot \underbrace{f(1 \dots x_n)}_{f_{x_1}} + \bar{x}_1 \cdot \underbrace{f(0 \dots x_n)}_{f_{\bar{x}_1}}$$

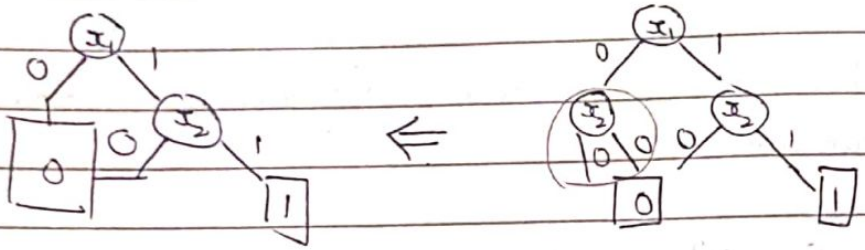
'Cofactor of f wrt x_1'

- Reduces one variable and reduces complexity

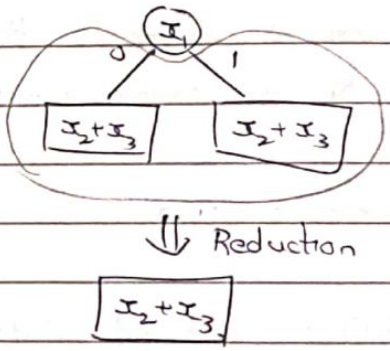


eg - $f = x_1 x_2$





eg $f = x_1(x_2 + x_3) + \bar{x}_1 x_2 + \bar{x}_1 x_3$



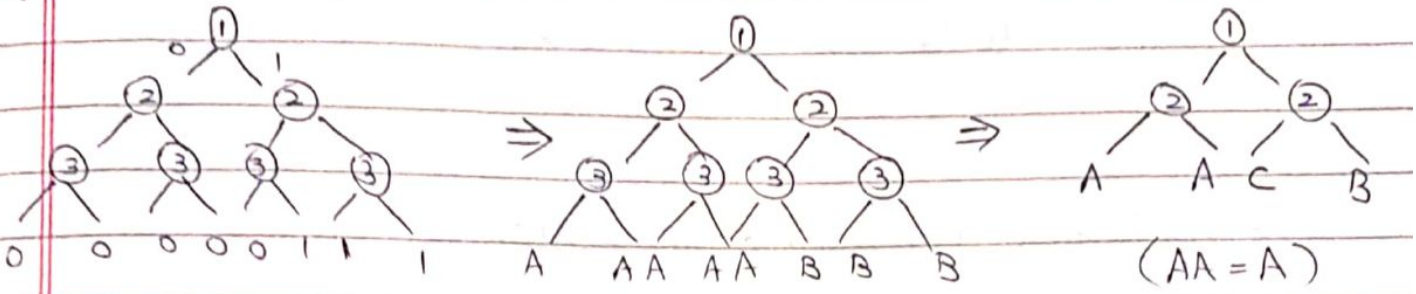
- There is a fixed order in which functions are checked : OBDD

Theorem Canonicity : ROBDD is for a function is unique, irrespective of order of variables, and order of reductions.

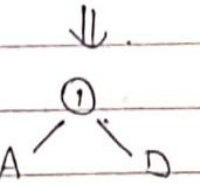
→ Reduction Algorithm.

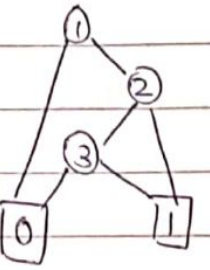
Start from leaves of complete graph. Assign label A to all zeros and B to all ones.

eg $f = x_1(x_2 + x_3)$



Node	Label
0	A
1	B
AB	C
CB	D
AD	E



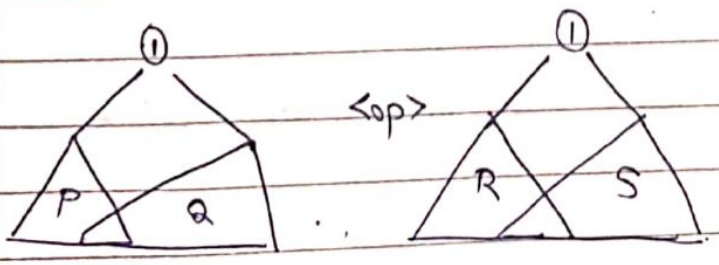


31/1

→ eg BDD for operation : $f \langle op \rangle g$ .... See paper

$$f(x_1 \dots x_n)\ op\ g(x_1 \dots x_n) = x_1\big(f(1 \dots x_n)\ op\ g(1 \dots x_n)\big) + \bar{x}_1\big(f(0 \dots x_n)\ op\ g(0 \dots x_n)\big)$$

(Q, S, P, R)



Using BDDs gives much lesser average time taken for operation

1)

* MUX operation ≡ If-then-else operation

$ite(x, y, z) = xy + \bar{x}z$

→ Substitution of a variable in a BDD.

→ ∴ There are functions for which BDD blows up.

eg - $x_1 x_{n+1} + x_2 x_{n+2} + x_3 x_{n+3} \cdots x_n x_{2n}$

- For verification of these functions, use other methods like satisfiability.

- Analysis:

$f = x_1 x_2 + x_3 x_4$

$g = x_1 x_4 + x_2 x_3$

BDD for f is smaller than BDD for g.

(Say, after computing x_2, BDD of f only needs to remember the product $x_3 \cdot x_4$, but BDD of g requires x_3 and x_4. Hence more memory required)

- Optimum ordering of variables is an unsolved problem.
- Heuristic: Decreasing order of no. of times the variable is use

- ~~In above example, for any order of variables, x_n's expression always blows up. ∴~~

(Appendix of paper)

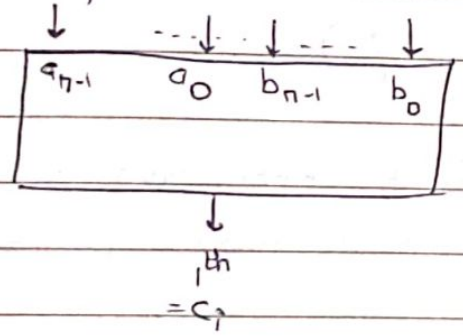
→ Apply Operation :- $f \langle op \rangle g$

$$= x_1 \left[f_{x_1} \langle op \rangle g_{x_1} \right] + \bar{x}_1 \left[f_{\bar{x}_1} \langle op \rangle g_{\bar{x}_1} \right]$$

If a function (say f) does not depend on x_1, its graph will not contain index '1'.
This just means that $f_{x_1} = f_{\bar{x}_1} = f$.

→ Multiplier

Appendix in paper

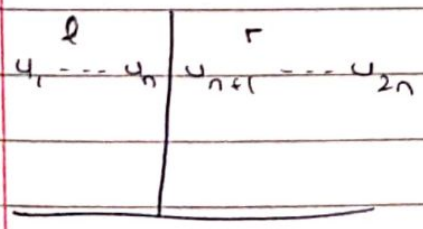


- Blows up

TPT: ∃ at least one i for which BDD of c_i blows up.

Jumble all a_i's and b_i's. Call them u_i's
↳ $i \in \{1 \dots 2n\}$

Divide them into equal partitions (left l and right r)

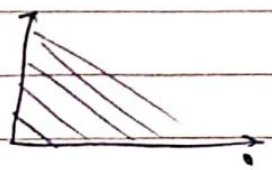


WLOG, l has $\geq \frac{n}{2}$ a_i's
r b_i's

There are $\geq \frac{n^2}{4}$ pairs of the kind (a_{i_l}, b_{i_r})

Consider $S_j = \{ (a_{i_l}, b_{i_r}) \} : a_{i_l} + b_{i_r} = j \}$

S_j plots lines $a_l + b_r = \text{const}$



j can go up to 2n.

P.T.O.

Hence there can be $2n$ no. of S_j's.

∴ We are dividing $\geq \frac{n^2}{4}$ pairs into $2n$ sets, there must be at least one S_j ~~for which~~ which contains at least $\frac{n^2/4}{2n} = \frac{n}{8}$ pairs.

(a_l, b_r)

$\frac{n}{8}$ a_i's on left | $\frac{n}{8}$ b_i's on right

(For each a_{i_l} ~~=~~ in this set, there exists a b_{i_r} from right set that makes $a_{i_l} + b_{i_r} = j$)

★ In the overall BDD, the different combinations of a_i's on the left set (total $\geq 2^{n/8}$) must produce different overall functions.

Hence, BDD blows up exponentially

→ Equivalence

① Check if $f_S = f_I$: Draw RoBDD - Compare. Should be same.

'Checking if equal'

② Check that, if $f_D = 1$, then $f_S = f_I$

i.e : $f_D \Rightarrow f_S = f_I$

i.e : $\overline{f}_D + (f_S \equiv f_I)$

Draw RoBDD for this. Should be '1' always

'Checking if always '1''

BDD Package

- Link libraries to main application
- Make BDD Manager — bdd_init()
  - ↳ universe for all BDDs
- Create variables in the BDDM, in any order.
  Can insert variables in required position
  - Each variable is also a BDD representing $f(x_1 \ldots x_n) = x_i$.
- Use operators to make BDDs
  - ↳ 2-input or 3-input or composition
    - ↓ ↳ if-then-else

- Compile : (see compile.sh)

gcc -o hw hw.c -I -L -lbdd -lmem

- man page :- cmu-bdd/man

- Traversing bdd :- bdd_if (root variable)
and bdd_else

SATISFIABILITY

eg Does there exist an x_1 for which $f(x_1 \ldots x_n) = 1$, for appropriate $x_2 \ldots x_n$?

i.e: '$\exists\, x_1$ s.t. $(f(x_1 \ldots x_n) == 1)$'

True or False?

This is called a property.

eg Property: $\exists x_1$ s.t $(f \cdot (x_2 \oplus x_3))$

- '$\exists\, x_1\ f(x_1, x_2, x_3)$' is a function of (x_2, x_3)

eg:-

$$f(x_1, x_2, x_3) = x_1 x_2 + \bar{x}_1 x_3$$

Then '$\exists\, x_1 f(x_1, x_2, x_3)$' $= x_2 + x_3$ .... Existential quantification function

eg:- $f(x_1, x_2, x_3) = x_1 x_2 x_3$

The EQF $= x_2 x_3$ .... if $x_2 x_3 = 1$, then $x_1 = 1$ exists for which $f = 1$.

- $\boxed{EQF = f_{x_1} + f_{\bar{x}_1}}$

- Different problem:- '$\forall\, x_1\ f(x_1, x_2, x_3)$'

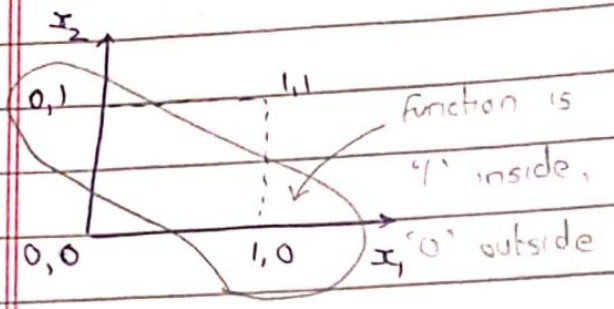
$$= \boxed{f_{x_1} \cdot f_{\bar{x}_1}}$$

'Universal Quantification function'

eg: $f = x_1 x_2 x_3 \longrightarrow UQF = 0$

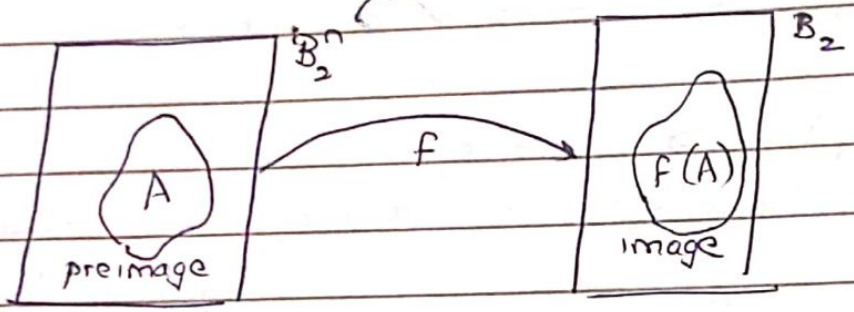
eg: $F = x_1 x_2 + \bar{x}_1 x_3 \longrightarrow UQF = x_2 x_3$

A] Symbolic Representation of sets



Represent it as $x_1 \oplus x_2$

eg.



eg:- f is a function that computes next state in an FSM.
$f(A) \equiv$ subset of B_2, the new reachable states from A
↓
subset of

* Type of problems
① Equivalence
② Satisfiability
③ Finding image of f given pre-image, or उल्टा

→ Characteristic Function of a function

For $y = f(x_1 \ldots x_n)$

or $X_f(x_1 \ldots x_n, y) = \overline{y_1 \oplus f(x_1 \ldots x_n)}$

.... '1' only when $y = f$.

- CF is a new function in 'n+1' variables. There need not be any distinction made between which is inputs and which is output.

→ CF of a vector of functions

$$y_i = f_i(x_1 \dots x_n)$$
$$X_i = \overline{y_i \oplus f_i(x_1 \dots x_n)}$$

Net CF = $X = X_1 X_2 \dots X_n$

→

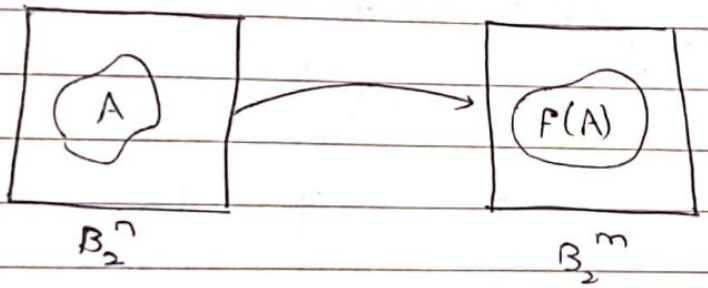


Image of function

Given :- Formula for A (evaluates to '1' iff element is in A) and function $y_1 \dots y_m = f(x_1 \dots x_n)$

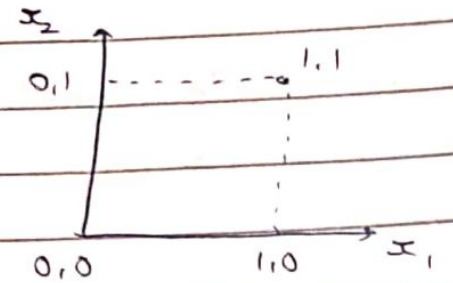
To find :- Formula for f(A)

Solution :- $\exists_{x_1 \dots x_n} [X_A X_f(x_1 \dots x_n, y_1 \dots y_m)]$ ← function of $(x_1 \dots x_n)$

∴ evaluates to '1' only for elements in f(A).

P.T.O.

eg



Consider $A \equiv \{(0,1)\}$

$f(x_1, x_2) = x_1 \oplus x_2$

$$X_f = \overline{y \oplus (x_1 \oplus x_2)}$$

$$X_A = \bar{x}_1 x_2$$

$$X_{F(A)} = \exists_{x_1, x_2} [X_A X_f]$$

$$= \exists_{x_2} \exists_{x_1} \left[\bar{x}_1 x_2 \left(\overline{y \oplus (x_1 \oplus x_2)} \right) \right]$$

$x_1 = 0$ ↙ ↘ $x_1 = 1$

$$= \exists_{x_2} \left[x_2 \left(\overline{y \oplus x_2} \right) + 0 \right]$$

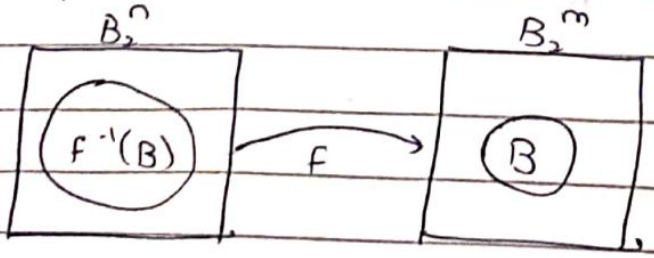
$x_2 = 0$ ↙ ↘ $x_2 = 1$

$$= [\, 0 + y \,]$$

$$= y$$

$\therefore \quad F(A) \equiv \{1\}$

→ Pre-image of function



$$X_{f^{-1}(B)} = \exists_{y_1 \cdots y_m} X_B(y_1 \cdots y_m) \, X_f(x_1 \cdots x_n, y_1 \cdots y_m)$$

* State Machine

Representation :- ① Next-state function $\delta : Q \times X \rightarrow Q$

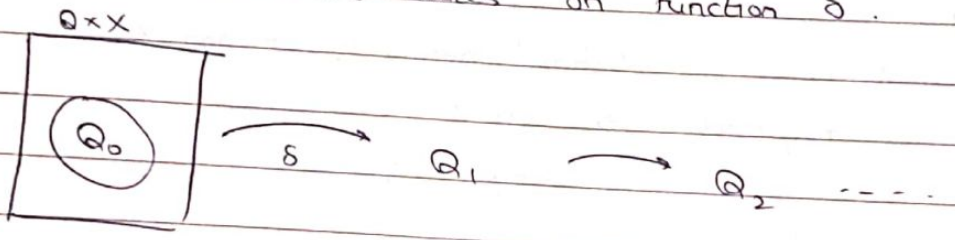
(states, inputs)

② Output function $\lambda : Q \times X \rightarrow Y$

(outputs)

- Reachability problem :- Is state j reachable from state i?

Starting from state 'i', keep calculating successive images of the set of reachable states on function δ.



$R_0 = Q_0$

$R_i = Q_0 + Q_1 \dots + Q_i \dots$ set of states reachable in 'i' steps.

Eventually, R_i will converge. (because no of states is finite)

$\downarrow$

$R_n = R_{n-1}$

* All these computations can be done with BDDs, but only if they don't blow up :-

I SAT - BASED METHOD

- Give up canonicity
- As an alternative to BDDs, in case they blow up.

eg Function expressed as (POS) → 'Conjunctive Normal Form'

$$x_1 \oplus x_2 = \underbrace{(x_1 + x_2)(\bar{x}_1 + \bar{x}_2)}_{\text{'Clause'}}$$

* Field :-

eg: Real numbers :- Defined operations + and ×.

$$x, y \in \mathbb{R} \Rightarrow xy \in \mathbb{R}$$
$$x+y \in \mathbb{R}$$

There exist a '0' and a '1'

eg $\{0, 1\}$ is a field with operations $\oplus$ and $\cdot$.

eg $f = \ldots PoS \ldots$

Each sum forms one equation

$$x_1 \oplus x_2 \oplus x_3 = 1$$
$$x_1 \oplus x_4 = 1$$
$$\vdots$$

This problem can be solved by matrix equations $Ax = 1$.

∴ Not all functions can be represented using ~~PoS~~. Product of XO

eg In CNF : $f(x_1 \dots x_n) = \underbrace{(x_1 + x_2 + x_3)}_{C_1} \dots\dots \underbrace{(\bar{x}_4 + x_5)}_{C_m}$

Solve simultaneous equations :- $C_i = 1 \quad \forall\, i$.

- In general, this is an NP complete problem :

- No. of bits required to describe the problem ~ $O(mn \log n)$
  (m: No. of clauses) ☆

- ∴ Time complexity ~ $O(2^{mn \log n})$
  (Searching all bit combinations : brute force)

→ 2-SAT problem

Each clause can have at most 2 literals

eg $(x_1 + x_2) \dots\dots (\bar{x}_4 + x_5)$

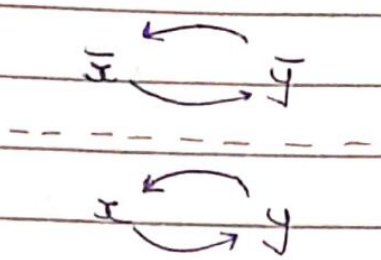
- Polynomial time algorithm exists :)

eg XNOR : $F = (x + \bar{y})(\bar{x} + y)$

* $(x + \bar{y} = 1) \iff (\bar{x} \Rightarrow \bar{y}) \cdot (y \Rightarrow x)$
* $(\bar{x} + y = 1) \iff (x \Rightarrow y) \cdot (\bar{y} \Rightarrow \bar{x})$

After generating all implications, draw the implication graph.

P.T.O.



In the graph, ~~form~~ find strongly connected components.

For each SCC, assign value to variables.

- This cannot be done if that SCC has edge from x to $\bar{x}$ and $\bar{x}$ to x

- Entire algo: Polynomial time.

→ XOR-SAT Problem.

Clauses are products of XORs of variables

- Can be solved by a method of Gaussian elimination
  - on a different field, not real numbers
    - 'Integers modulo 2'

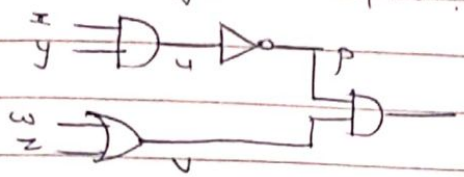
Express as $Ax = b$.

∴ Not all functions can be expressed as product of XORs

→ 3-SAT Problem

→ Generating CNF

eg Start with gate-description.



$u \equiv xy$
$p \equiv \bar{u}$
$v \equiv \omega + z$
$q \equiv pv$

} Intermediate clause

* $a \equiv b$ is equivalent to $(a \Rightarrow b) \cdot (b \Rightarrow a)$

For AND gate, $u \equiv (xy)$

$= (u \Rightarrow xy)(xy \Rightarrow u)$

$= (\bar{u} + xy)(\bar{x}\bar{y} + u)$

$= (\bar{u} + x)(\bar{u} + y)(\bar{x} + \bar{y} + u)$ .... 3 clauses

OR gate ~ 3 clauses, NOT gate ~ 2 clauses.

Hence, no. of clauses ~ O (polynomial in no. of gates) ☺

However, have to introduce new (internal) variables ☹

- If CNF for R, Q are known, find CNF for P.

① $P = QR$

Easy :- Product of clauses of Q & clauses of R
$- \{$common clauses$\}$

② $P = \bar{Q}$ } Difficult. Will need to use distributivity of
③ $P = Q + R$ } + over · to convert from SoP to PoS.
May produce exponential no. of clauses.

- Once CNF is known :- 2 algorithms of interest.

① DPLL

② GRASP

A] DPLL Algorithm

- If a clause has a single literal (eg : $C_1 = x_1$), simply assign value and eliminate that variable ($x_1 = 1$)

- If no clause contains, say, $\bar{x}_1$, simply assign $x_1 = 1$ to eliminate x_1.

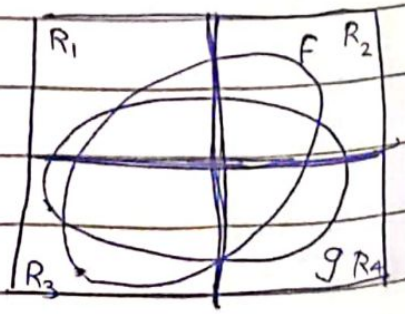
- With the remaining variables, explore the search space. Start assigning values to variables ~~in~~ one by one in a particular order, and then check the newly obtained function recursively for satisfiability.

Heuristic :- Most-occurring variable first

In worst case, one would have to explore the entire tree by assigning all values to all variables.

✳ Verification by dividing search space into regions

To verify :- $f \equiv g$



Check in R_1 : ~~R_1~~

Check satisfiability for $R_1 \Rightarrow (f \equiv g)$

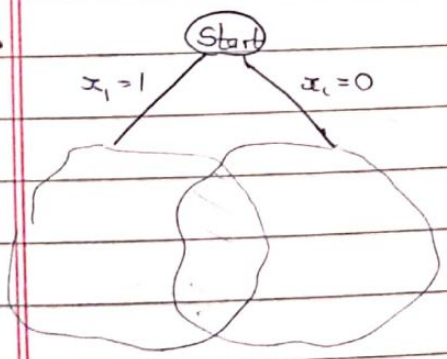
* Minisat Solver Package

- Starts with c : comment
  p : problem.
  eg: p cnf 4 5 : 4 variables, 5 clauses
  0 is delimiter for clauses
  -x means $\bar{x}$.

- Output file gives satisfying set.

- How to generate CNF? : Manual for now

B] GRASP

- 

Some clauses might not depend on x_1.
Need to learn from every branch to ~~Suppose we realize~~ minimize computation.

Optimization: Suppose we realize that $x_1, x_2, x_3, x_4 = 1111$ does not give solution, we add a clause $(\bar{x_1} + \bar{x_2} + \bar{x_3} + \bar{x_4})$ to the problem.

☆ Computation time might be reduced when we try to run the Kya faayda new problem again

Optimization 2 Non-chronological back-tracking

In normal DPLL, we do chronological backtracking. If $x_1 = 1$ does not ~~do~~ satisfy, we proceed with $x_1 = 0$. (We do depth-first search)

* $x_5 = 1 @ 9$

[x_5 was made '1' at decision #9]

☆ eg Implication graph at decision #6. for x_1

Verify this shit

$\omega_1 = -x_1 + x_2$

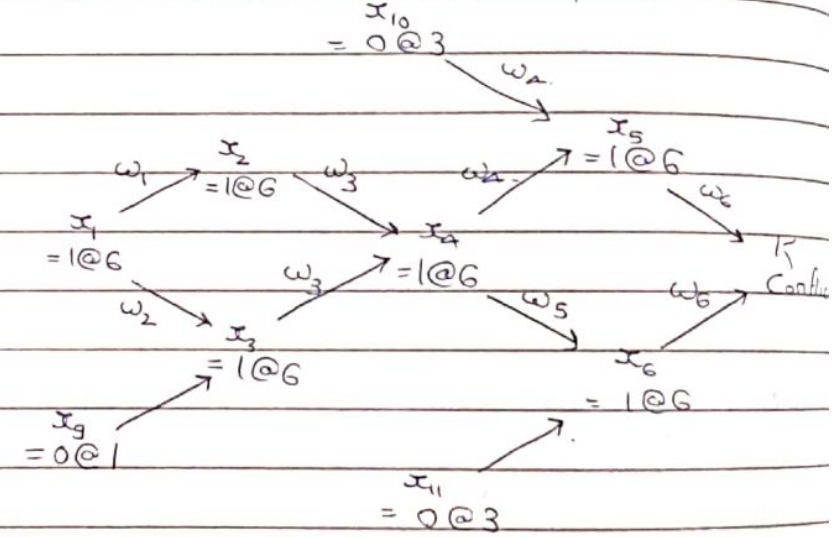
$\omega_2 = -x_1 + x_3 + x_9$

$\omega_3 = -x_2 + -x_3 + x_4$

$\omega_4 = -x_4 + x_{10} + x_5$

$\omega_5 = -x_4 + x_6$

$\omega_6 = -x_5 + -x_6$



Since $x_1 = 1@6$ gave us conflict, we will try $x_1 = 0@6$

Conflict $x_1 = 1@6$ was caused by the roots of above graph : x_9, x_{10}, x_{11}, x_1

☆ So we add a clause $(\bar{x}_1 + x_9 + x_{10} + x_{11})$

Suppose $x_1 = 0@6$ also caused conflict and the roots of that new graph were : x_{13}, x_1

$x_{13} = 1@4$

☆ Add $(x_1 + \bar{x}_{13})$

Since no value of x_1 @6 can be a solution, we need to backtrack.

Backtrack to a decision at the largest step number that had caused the conflicts at this decision @6.

- In this case, go to x_{13} @4: Must try both values of x_{13} again, now with the new clauses. ★
  (this will help detect conflicts sooner.)