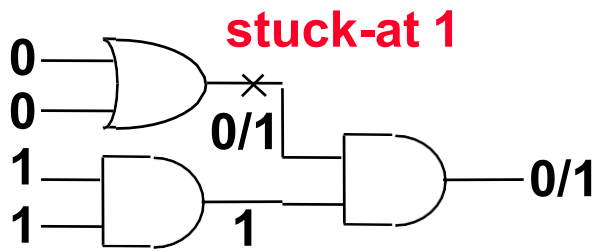
Combinational Test Generation

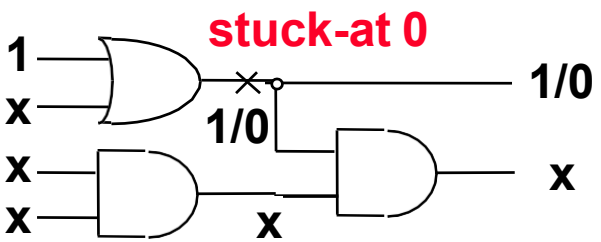
- **Test Generation (TG) Methods**
  - **(1) From truth table (2) Using Boolean equation (3) Using Boolean difference (4) From circuit structure**
- **TG from Circuit Structure**
  - **Common Concepts**
  - **ATPG Algorithms : D-Algorithm**

A Test Pattern

- A test pattern



- A test pattern with don't cares



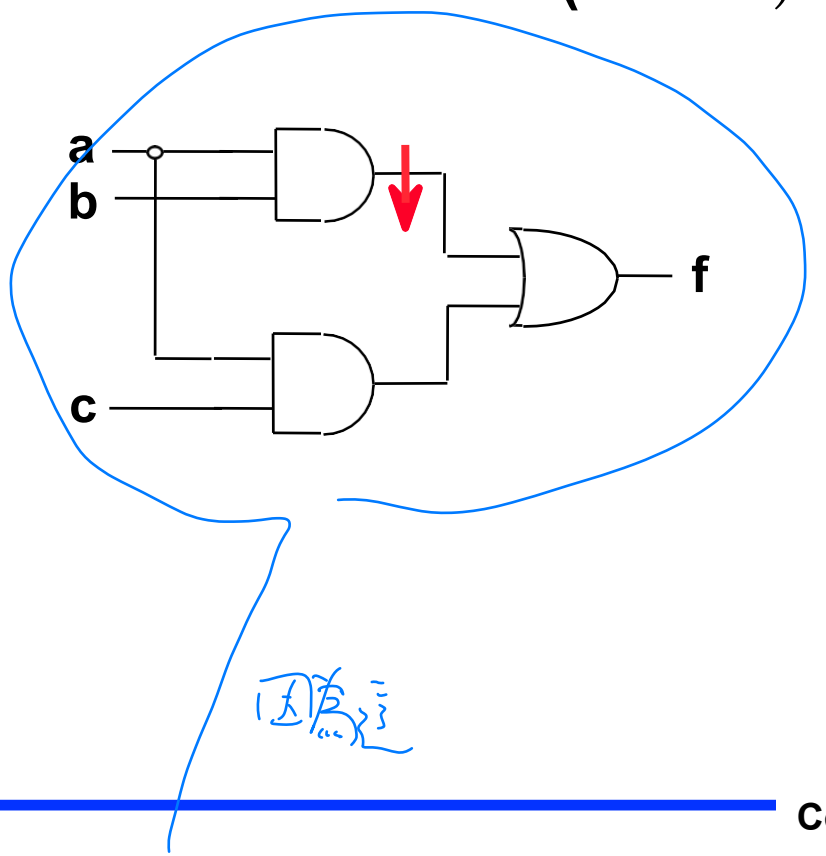
(Good value and faulty value are different at a PO.)

- Test generation: generates a test for a target fault.

生成 可以測 目標錯誤 的輸入

Test Generation Methods
(From Truth Table)

Ex: How to generate tests for the stuck-at 0 fault (fault α)?



	abc	f	f_α
	000	0	0
	001	0	0
	010	0	0
	011	0	0
	100	0	0
	101	1	1
√	110	1	0
	111	1	1

Impractical !!

所有都測過一遍
是不實際的 TLE

Test Generation Methods
(Using Boolean Equation)

用布林 方程式 推出

Since f = ab+ac, f_α = ac =>

T_α = the set of all tests for fault α

= ON_set(f) * OFF_set(f_α) + OFF_set(f) * ON_set(f_α)

= {(a,b,c) | (ab+ac)(ac)' + (ab+ac)'(ac) = 1}

= {(a,b,c) | abc'=1}

= { (110) }.

太複雜

High complexity !!

Since it needs to compute the faulty function for each fault.

所有 能讓 f輸出 為1的 輸入組合 就叫ON set

- ***ON_set(f): All input combinations that make f have value 1.***

OFF_set(f): All input combinations that make f have value 0.

布林 差分式

Boolean Difference

目的也是為了 找到某個測試訊號輸入
能讓想測的地方 的結果 傳遞到輸出

- **Physical Meaning of Boolean Difference**
  - For a logic function $F(X)=F(x_1, ..., x_i, ..., x_n)$, find all the input combinations that make the change of value in x_i also cause the change of value in F.
- **Logic Operation of Boolean Difference**
  - The Boolean difference of F(X) w.r.t. input x_i is

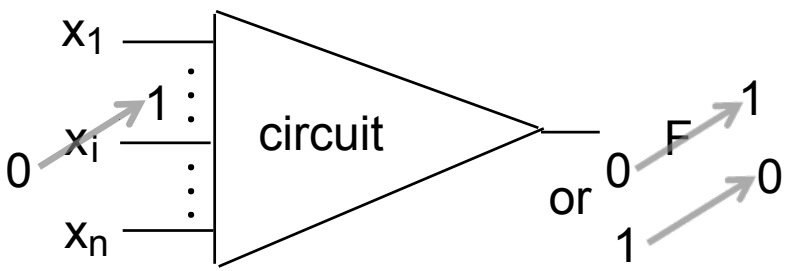
$$\frac{dF(X)}{dxi} = F_i(0) \oplus F_i(1) = \overline{F_i(0)} \bullet F_i(1) + F_i(0) \bullet \overline{F_i(1)},$$

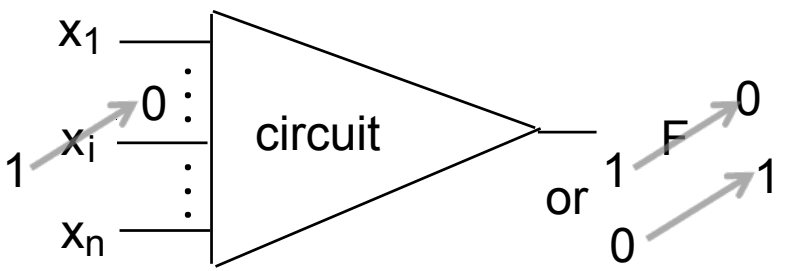
這裡 微分 代表 0變1 1變0

會要用這個XOR
是為了 希望在輸入 0 和 1 會有不同的結果
所有這個正常結果 是1

where $F_i(0) = F(x_1, ..., 0, ..., x_n)$ and $F_i(1) = F(x_1, ..., 1, ..., x_n)$.

- **Relationship between TG and Boolean Difference**

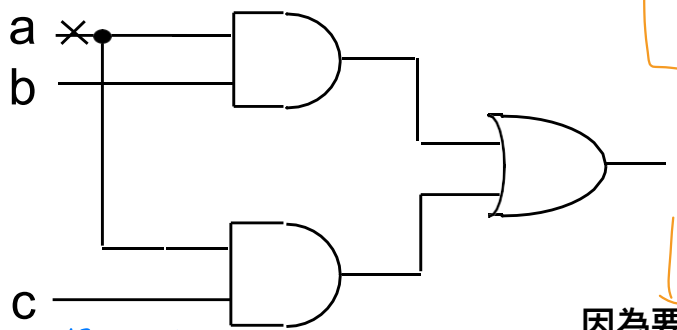




目的是要找到測試資料

Applying Boolean Difference to Test Generation (1/2)

Case 1: Faults are present at PIs.



f: 整个电路输出

f = ab+ac => $\frac{df}{da} = f_a(0) \oplus f_a(1) = 1 \bullet (b+c)+0 = b+c$

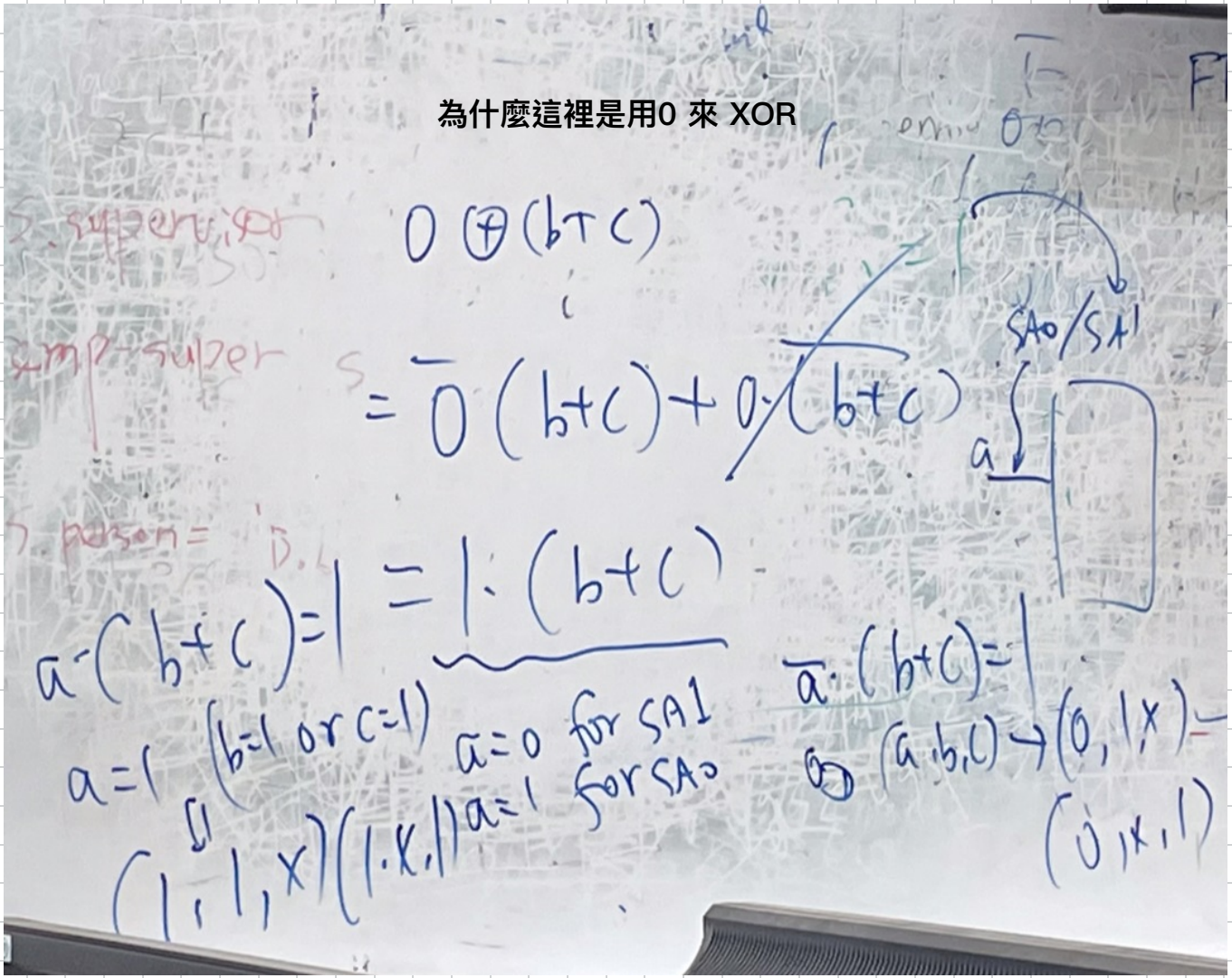
a代入0 和 a代入1做XOR

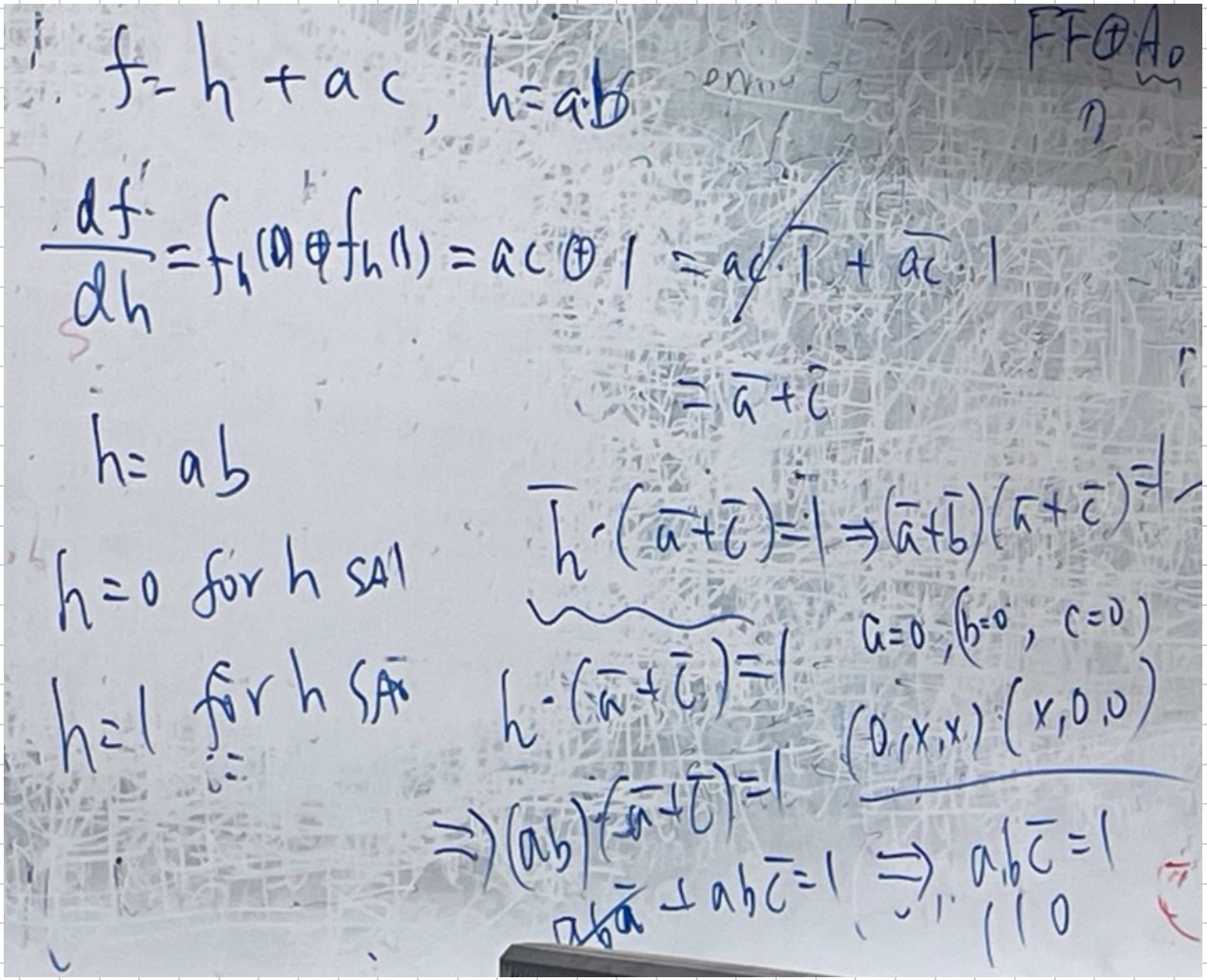
因為要測試

The set of all tests for line *a* s-a-1 is {(a,b,c) | a'*(b+c)=1} = {(01x), (0x1)}.
The set of all tests for line *a* s-a-0 is {(a,b,c) | a*(b+c)=1} = {(11x), (1x1)}.

並且想 input 1和0 XOR是1 → 用0測 ⇒ a
且分別去看那个 input 卡在1 的 test有誰 ⇒再加上一个條件，and起來
卡在0 → 用1測 ⇒ a

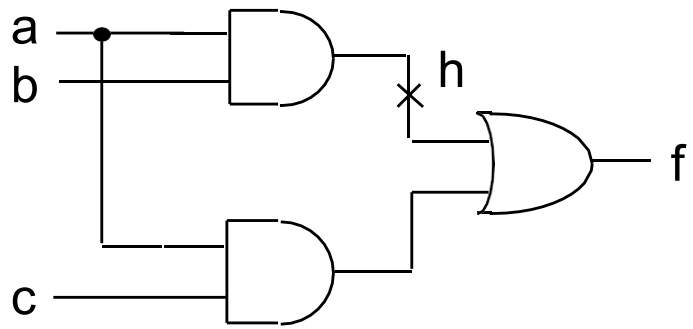
為什麼這裡是用0 來 XOR




f = h + ac, h = ab
FFOA0
df/dh = f_h(0) ⊕ f_h(1) = ac ⊕ 1 = ac·1 + ac̄·1
= ā + c̄
h = ab
h = 0 for h SA1
h = 1 for h SA
h̄·(ā + c̄) = 1 ⇒ (ā + b̄)(ā + c̄) = 1
h·(ā + c̄) = 1
(ab)(ā + c̄) = 1
a=0, (b=0, c=0)
(0,x,x) (x,0,0)
abc̄ = 1 ⇒ a,b c̄ = 1
110

Applying Boolean Difference to Test Generation (2/2)

Case 2: Faults are present at internal lines.



ac ⊕ l

f = h+ac, h = ab => $\frac{df}{dh} = f_h(0) \oplus f_h(1) = \overline{ac} \bullet 1 + ac \bullet \overline{1} = \bar{a} + \bar{c}$

$\bar{h} = \overline{ab} = \bar{a} + \bar{b}$

The set of all tests for line *h* s-a-1 is

H卡1 這樣就會有兩種通解

{ (a,b,c)|h'∗(a'+c')=1 } = { (a,b,c)|(a'+b')∗(a'+c')=1 } = { (0xx), (x00) }.

The set of all tests for line *h* s-a-0 is

{(a,b,c)|h∗(a'+c')=1} = {(110)}.

h卡0 這樣只會有一個解

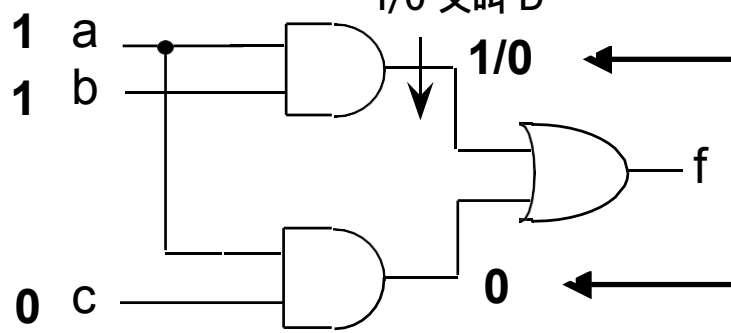
ab'

這裡就是要從 要測試的地方 推到輸出 能夠穿出去
再反推到 其他一開始的輸入

Test Generation Methods

(From Circuit Structure)

向下 叫 卡在0
用1去測
1/0 又叫 D



測試錯誤 就要用反向的訊號 測試

fault activation

錯誤傳遞

fault propagation

- **Two basic goals:**

找 test set

Fault activation(FA)

這兩個合稱 又叫

=> Line justification (LJ)

傳到最後面

Fault propagation(FP)

1/0 代表 正常是1 錯誤是0 這樣給他一個符號 D
0/1 同理反推 D'

where 1/0 means that the good value is 1 and the faulty value is 0 and is denoted as D. Similarly, 0/1 is denoted as D'.
D and D' are called fault effects (FE).

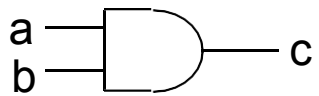
錯誤效益

Common Concepts for Structural TG

- The FA problem => a LJ problem.
- The FP problem =>
  (1) Select a FP path to a PO => decisions.
  (2) Once the path is selected => a set of LJ problems.
- The LJ problems => decisions or implications .

  決定 暗示

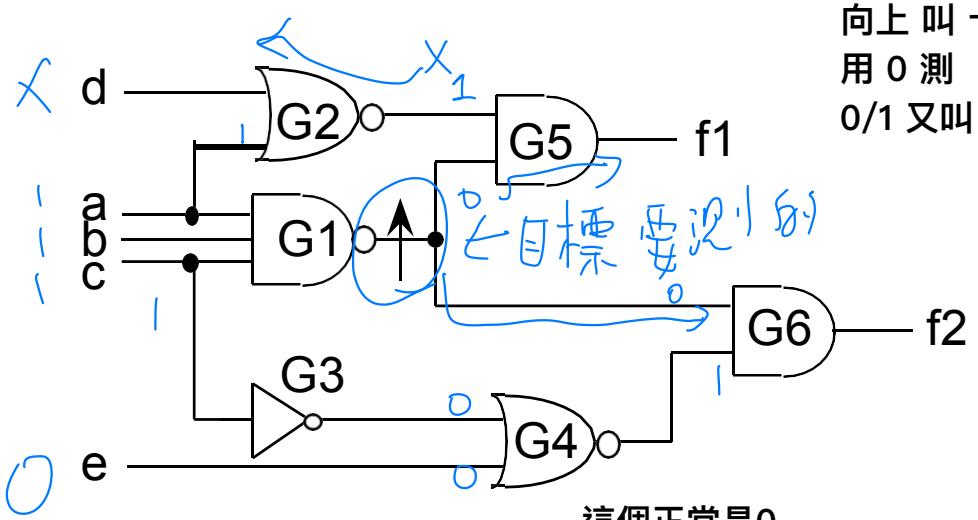
  ex:



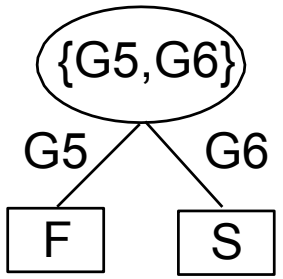
  To justify c=1 => a=1 and b=1. (implication) 有c是1 就 暗示 a b都是 1
  To justify c=0 => a=0 or b=0. (need make decisions) 有c是0 就需要 決定 a b 誰是0 或都是
- Incorrect decision => Backtracking => Another decision.
- Once the fault effect is propagated to a PO and all line values to be justified are justified, the test is generated. Otherwise, the decision process must be continued repeatedly until all possible decisions have been tried.

錯誤效益 錯誤能夠傳送到 最後的輸出 代表這個可以當作 測試錯誤的訊號
如果都傳不出去 就要重新設計電路

Ex: Decisions When Fault Propagation



向上 叫 卡在1
用 0 測
0/1 又叫 D'



The corresponding decision tree

錯誤啟動

這個正常是0
0/1

錯誤傳遞　要能傳送到最後的輸出

FA => a=1, b=1, c=1 => G1= D', G3=0; FP => through G5 or G6.

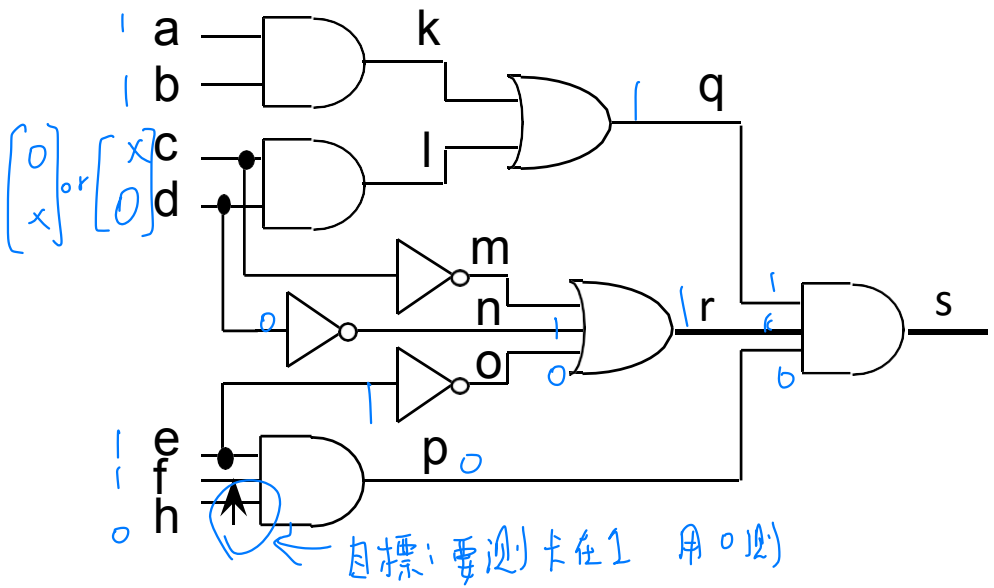
Decision: through G5 => G2=1 => d=0, a=0. => inconsistency => backtracking!!

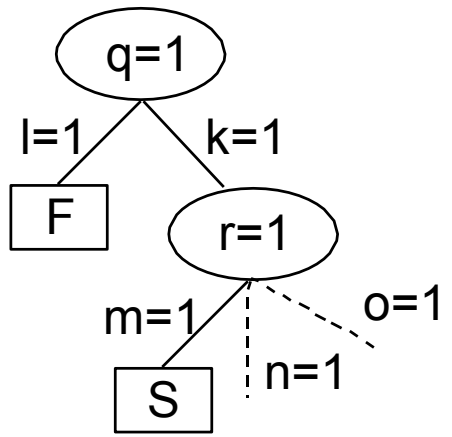
Decision: through G6 => G4=1 => e=0. => done!!

The resulting test is 111x0.

***D-frontier**: The set of all gates whose output value is currently x but have one or more fault signals on their inputs. Ex: Initially, the D-frontier of this example is {G5, G6}.*

Ex: Decisions When Line Justification





The corresponding decision tree

FA => h=D'; FP => e=1, f=1 (=> o=0); FP => q=1, r=1.
To justify q=1 => l=1 or k=1.
Decision: l=1 => c=1, d=1 => m=0, n=0 => r=0. => inconsistency => backtracking!!
Decision: k=1 => a=1, b=1.
To justify r=1 => m=1 or n=1 (=> c=0 or d=0). => done!!

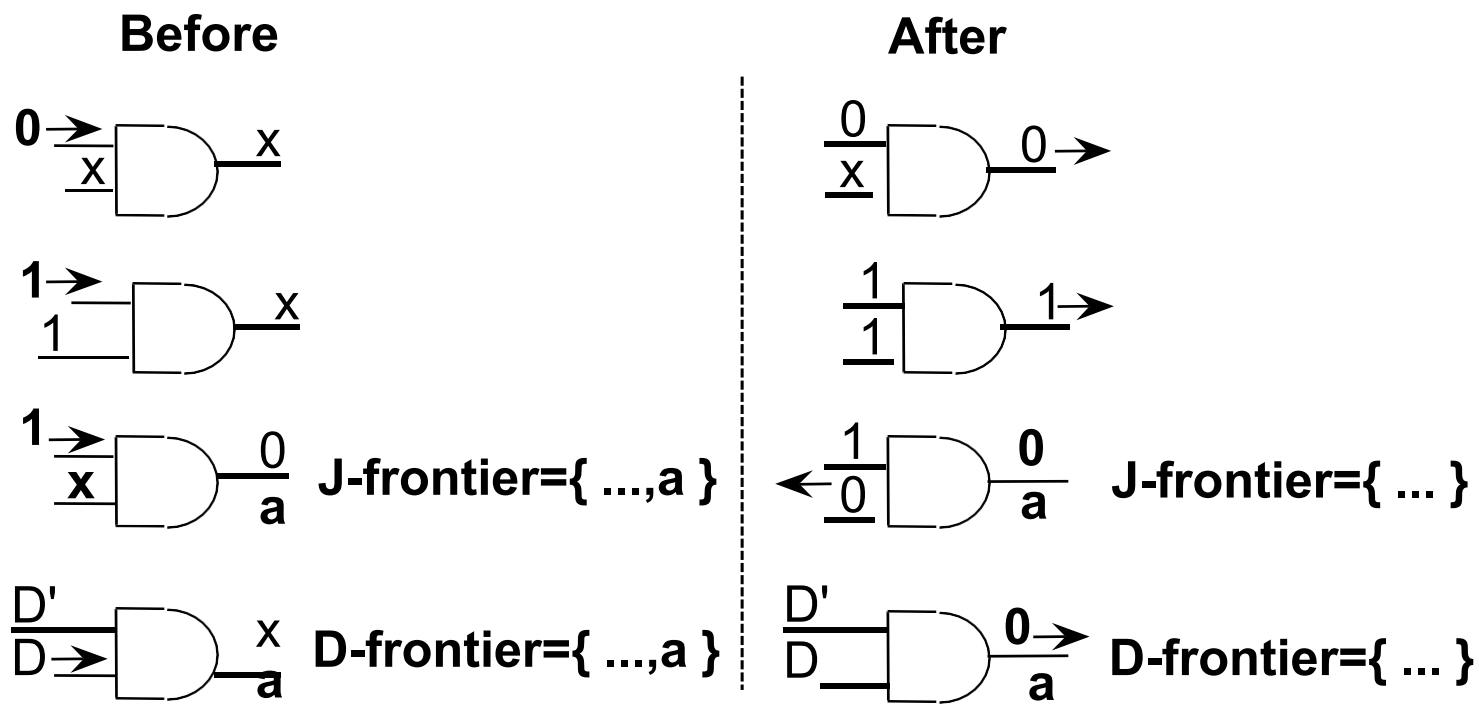
J-frontier: ***The set of all gates whose output value is known but is not implied by its input values. Ex: Initially, the J-frontier of the example is {q=1, r=1}.***

Implications

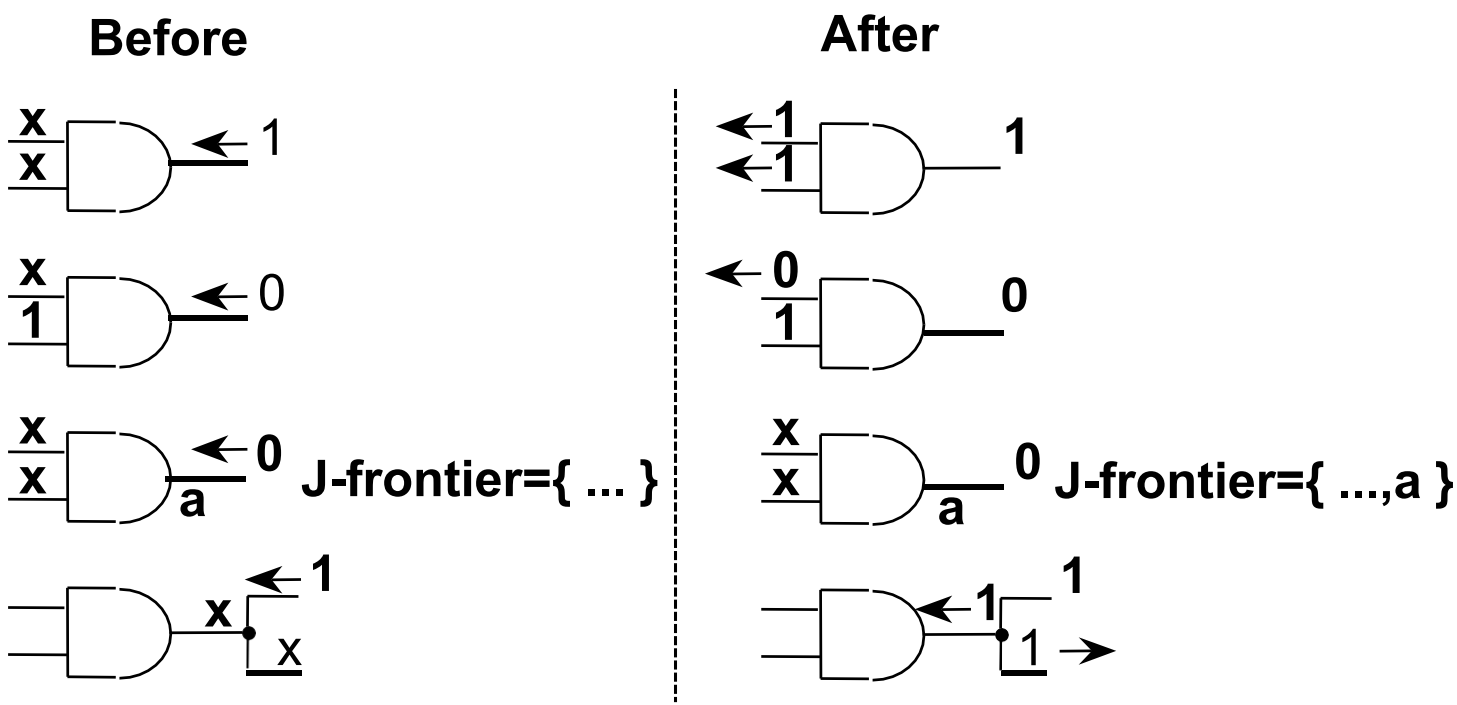
暗示 就是 一個地方決定了 某一個地方 就一定能直接推斷而出

- **Implication: computation of the values that can be uniquely determined.**
  - **Local implication: propagation of values from one line to its immediate successors or predecessors.** 區域性的 暗示 能推斷出 一個後代 或 祖先
  - **Global implication: the propagation involving a larger area of the circuit and reconvergent fanout.** 全域性的 暗示 會傳遞 推斷到一組電路的某個地方 一定是某值
- **Maximum implication principle: perform as many implications as possible.** 能夠推的 都寫出來
- **Maximum implications help us to either reduce the number of problems that need decisions or to reach an inconsistency sooner.** 可以很快發現誰是矛盾的

Local Implications (Forward)

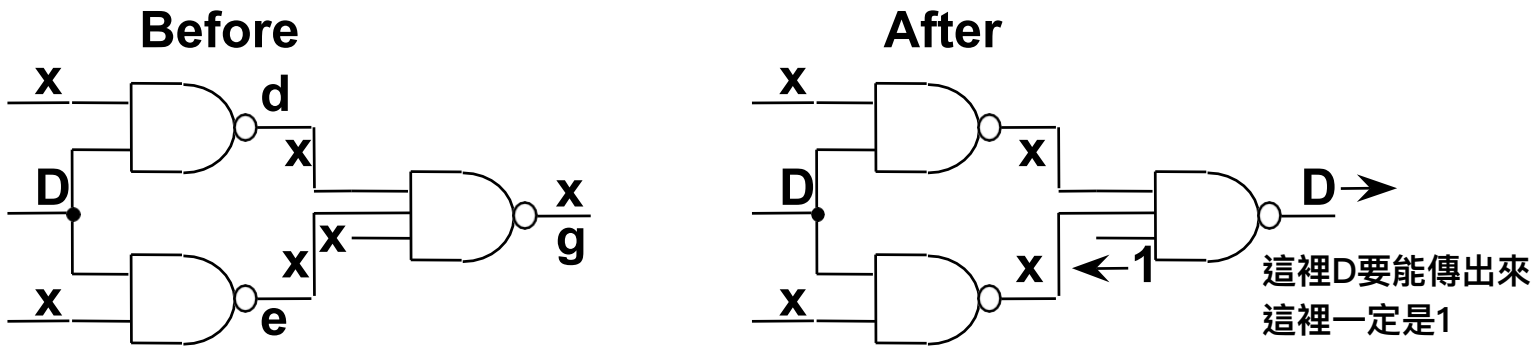


Local Implications (Backward)



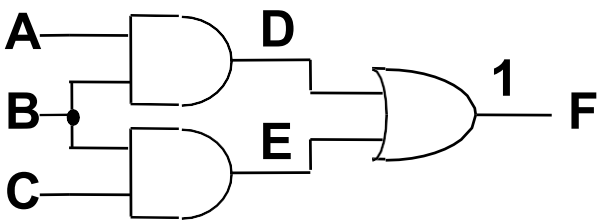
Global Implications

全域性 暗示



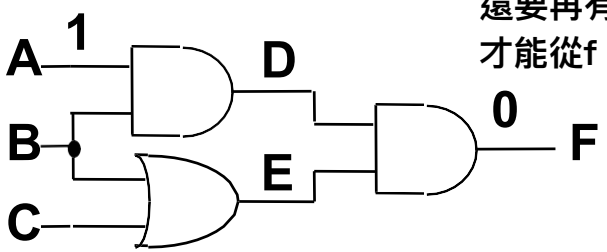
(1) Future unique D-drive.

靜態 推斷 f 1



(2) F=1 implies B=1.
(Static learning)

動態的 推斷
還要再有一個條件
才能從f 0 推斷B 要是 0



(3) F=0 implies B=0 when A=1.
(Dynamic learning)

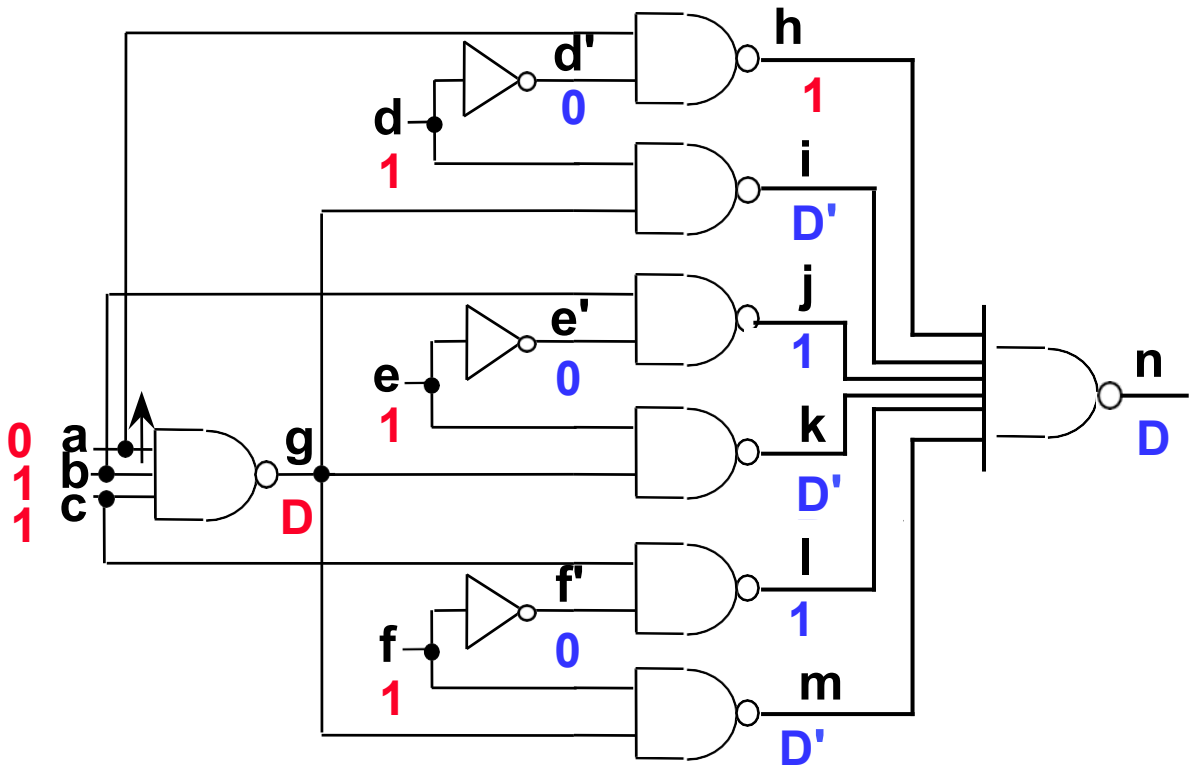
(2), (3) are based on contraposition law: (A=>B) <=> (!B => !A).

D-Algorithm: Example

也是和之前一樣 要測的地方 找路出去

能用的訊號 有這幾種

- Logic values = {0, 1, D, D', x}.

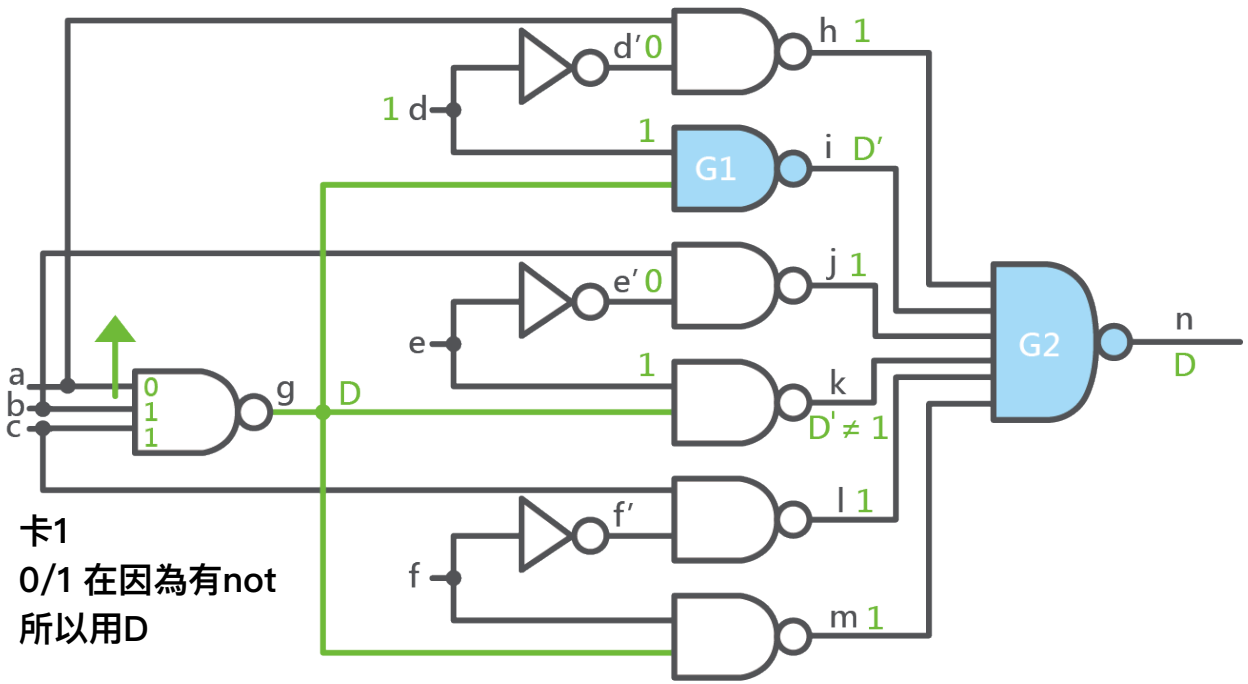


Assignment

Implication

D-Algorithm: Example

- **Logic values = {0, 1, D, D', x}.**



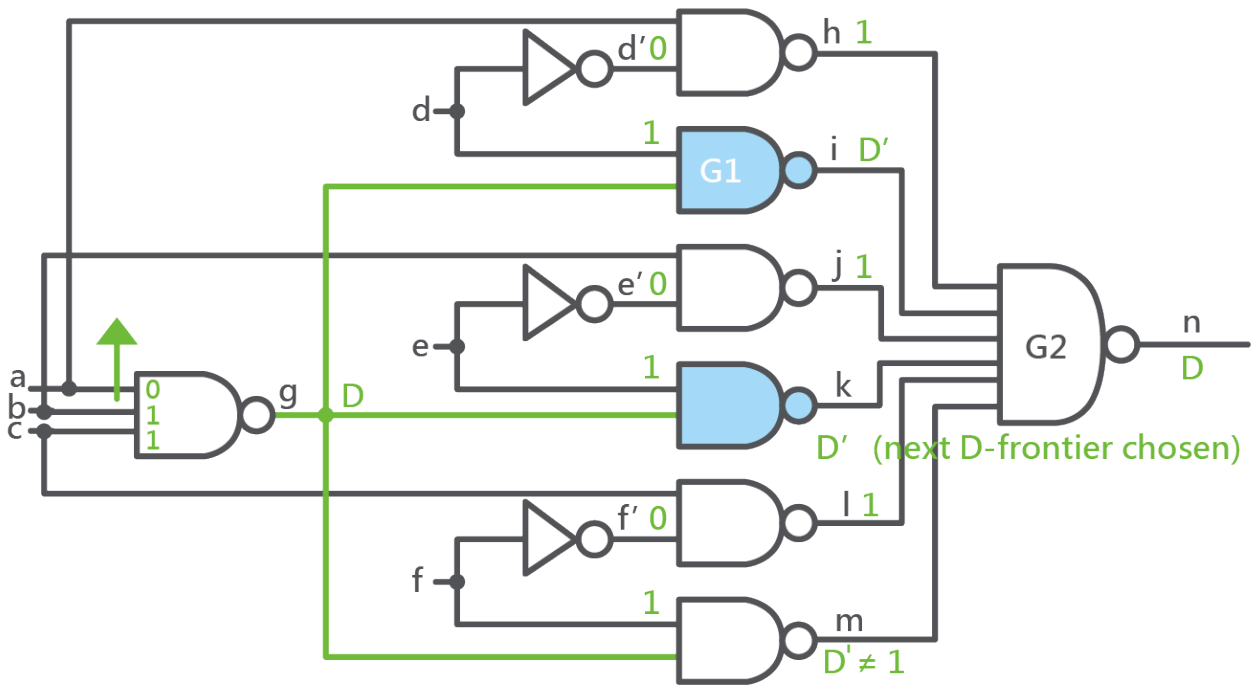
卡1
0/1 在因為有not
所以用D

想從G1 傳遞到最後
來去推
e那邊會有問題 k會錯

1. Propagate fault effect through G1 → Set d to 1
2. Propagate fault effect through G2 → Set j,k,l,m to 1
3. Conflict occured at k → Backtrack

D-Algorithm: Example

- **Logic values = {0, 1, D, D', x}.**

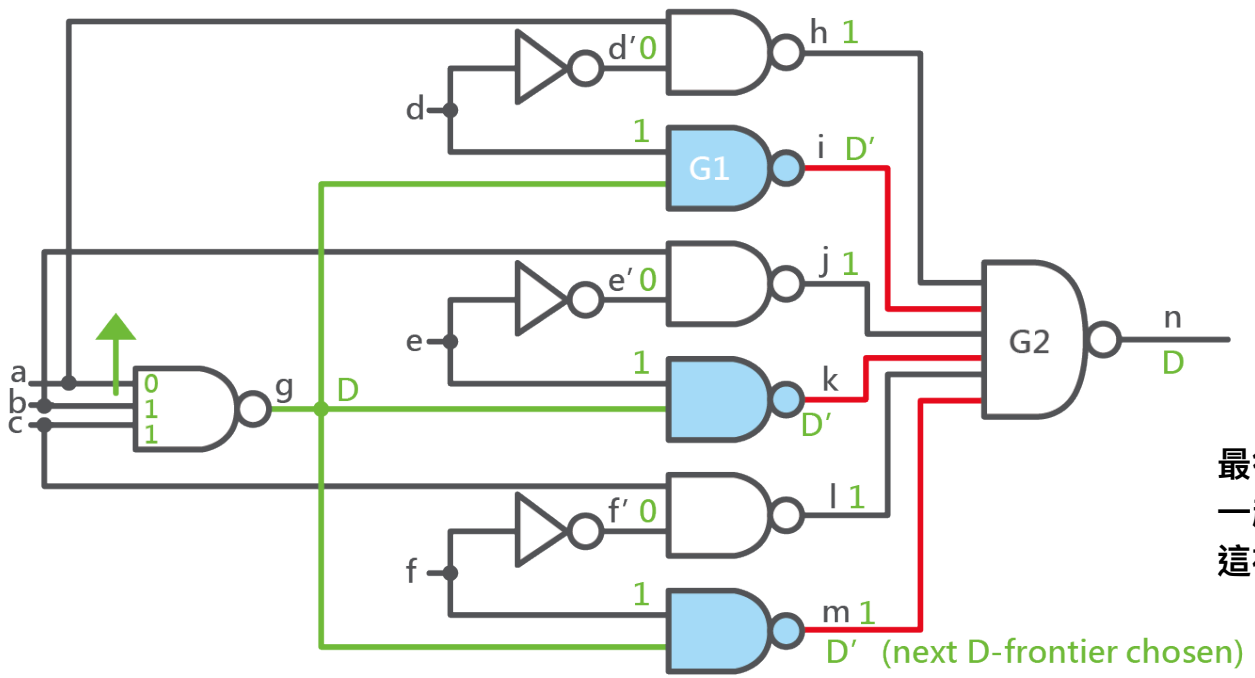


想從 G2 去傳遞到最後輸出
來去推斷
也有會有問題

1. Propagate fault effect through G2 → Set j,l,m to 1
2. Conflict occured at m → Backtrack

D-Algorithm: Example

- **Logic values = {0, 1, D, D', x}.**



自己再做一次

最後在最下面 才傳的出去

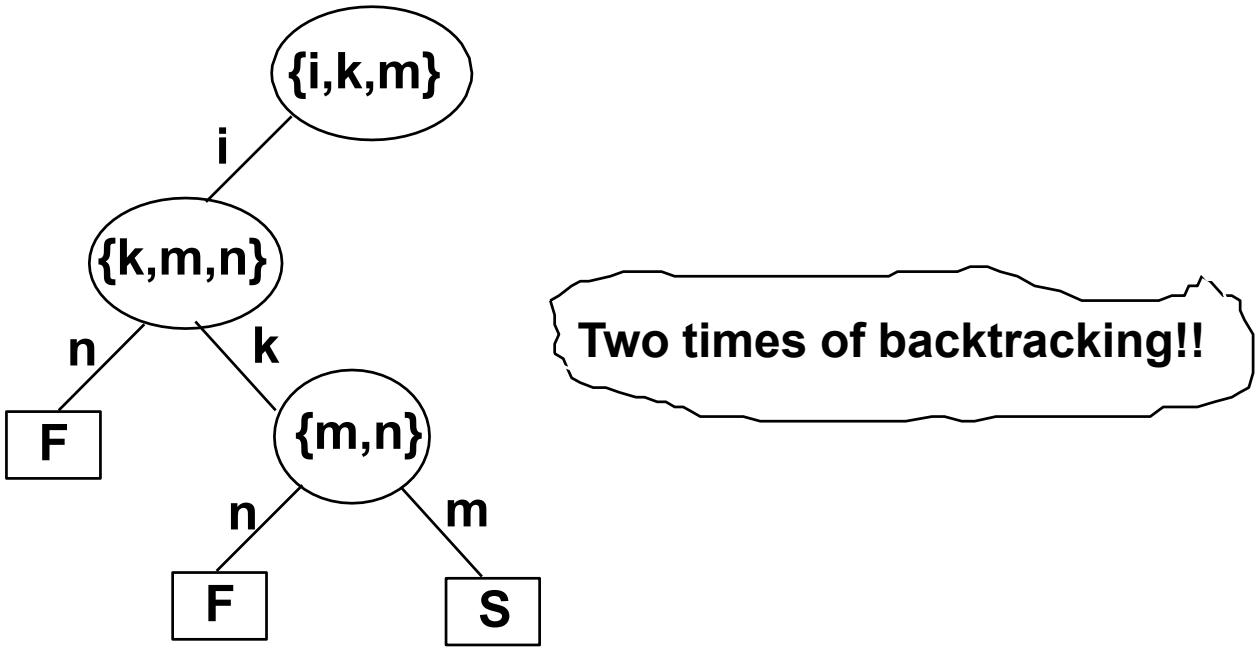
1. Propagate fault effect through G2 → Set j,l to 1
2. Fault propagation and line justification finish

最後結果就是 會同時走這三個地方一起出去
這裡有種 未知數 代數 x 的感覺

D-Algorithm: Value Computation

Decision	Implication	Comments
	a = 0 h = 1 b = 1 c = 1 g = D	Active the fault Unique D-drive
d = 1	i = D’ d’ = 0	Propagate via i
j = 1 k = 1 l = 1 m = 1	n = D e’ = 0 e = 1 k = D’	Propagate via n **Contradiction**
e = 1	k = D’ e’ = 0 j = 1	Propagate via k
l = 1 m = 1	n = D f’ = 0 f = 1 m = D’	Propagate via n **Contradiction**
f = 1	m = D’ f’ = 0 l = 1 n = D	Propagate via m

D-Algorithm: Decision Tree



- Decision node: the associated D-frontier.
  branch: the decision taken, i.e., the gate selected from the D-frontier.
- The D-algorithm first tried to propagate the fault solely through i, then through both i and k, and eventually succeeded when all three paths were simultaneously sensitized.

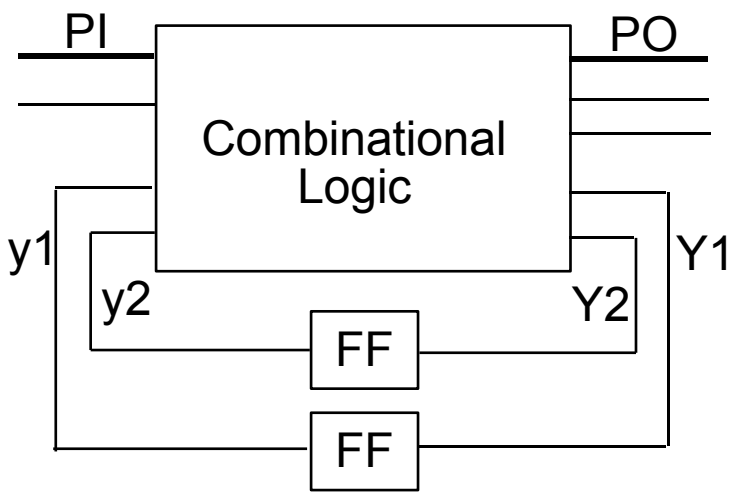
Iterative Logic Array (ILA) Model for Sequential Circuits

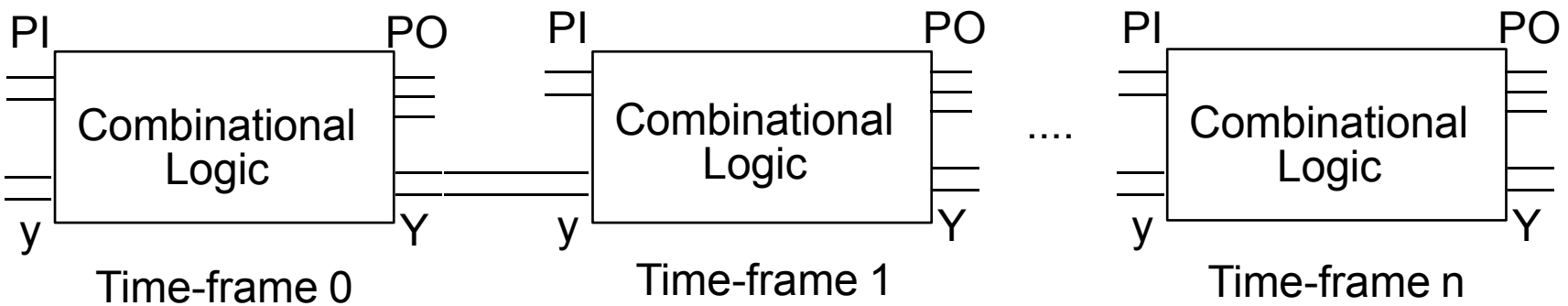
會儲存上一次的輸出
就叫做 續向邏輯
這次的輸出 會和上一次有關係

會比較難測

所以我們會想把它變成組合邏輯來測

把它想像成動畫
一秒鐘30個畫面
看他是第幾貞的畫面 訊號
多複製幾個 這一秒 和 上一秒 的自己 相連



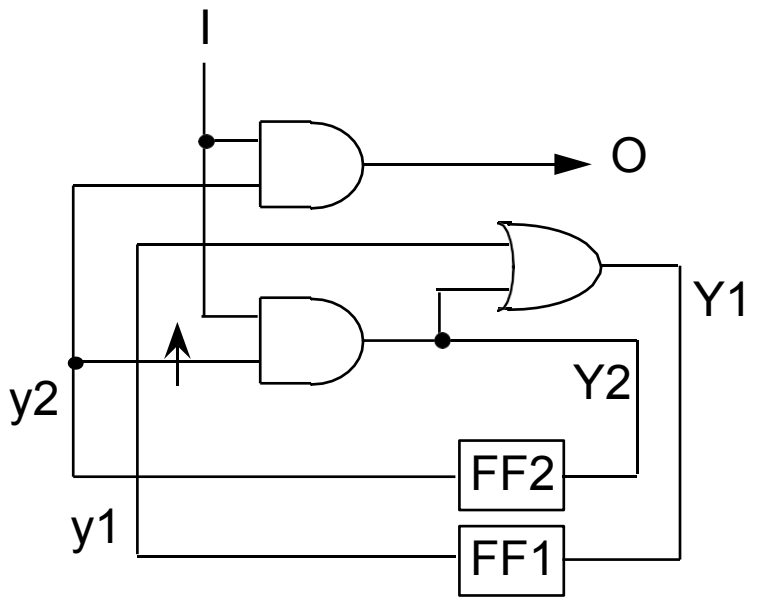


Sequential Test Generation

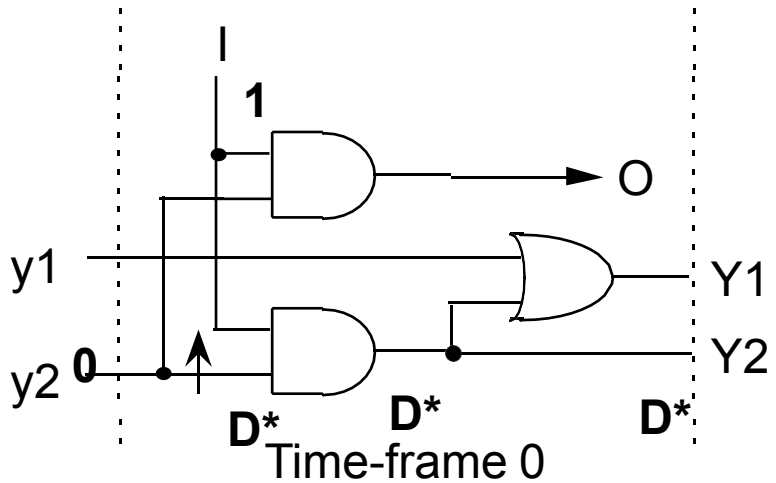
Extended D-Algorithm

1. Pick up a target fault f.
2. Create a copy of a combinational logic, set it time-frame 0.
3. Generate a test for f using D-algorithm for time-frame 0.
4. When the fault effect is propagate to the DFFs, continue fault propagation in the next time-frame.
5. When there are values required in the DFFs, continue the justification in the previous time-frame.

Example for Extended D-Algorithm

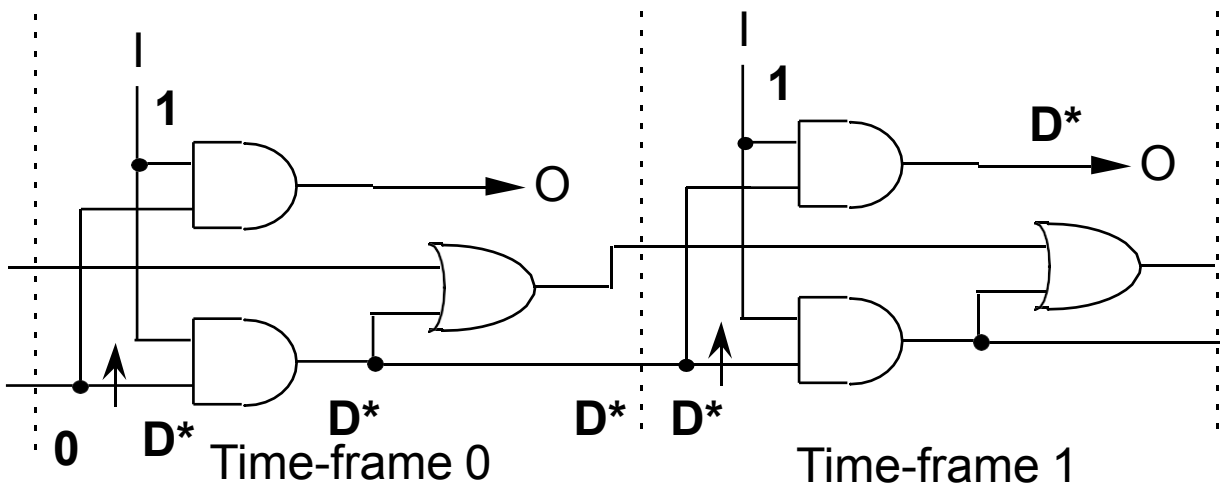


Example: Step 1



Example: Step 2

這樣 到 第二秒 D’ 就能傳遞到輸出



但是這裡的 0 也要從

Example: Step 3