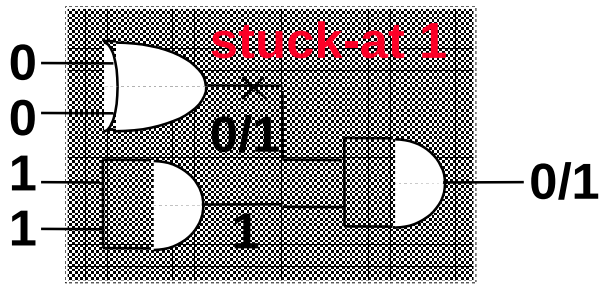
Combinational Test Generation

- **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**
  - **Algorithms : D-Algorithm (Roth 1967), 9-V Algorithm (Cha 1978), PODEM (Goel 1981), FAN (Fujiwara 1983), Socrates (Schultz 1987)**

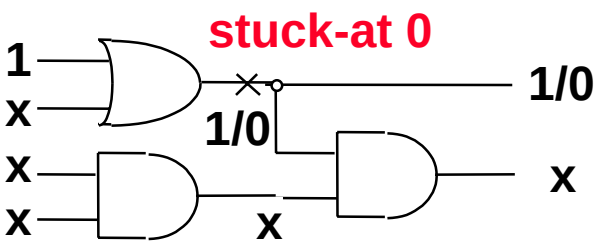
(Source: NCTU 李崇仁 教授)

A Test Pattern

- A test pattern



- A test pattern with don't cares

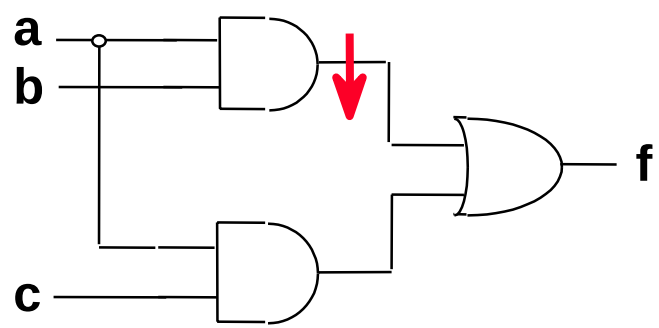


(Good value and faulty value are different at a PO.)

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

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 !!

Test Generation Methods
(Using Boolean Equation)

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) }.

High complexity !!

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

- ***ON_set(f): All input combinations that make f have value 1.***
  OFF_set(f): All input combinations that make f have value 0.

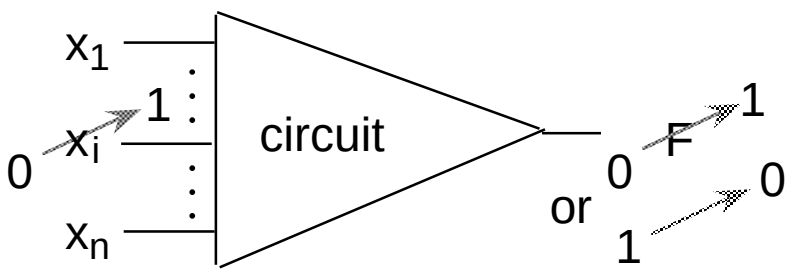
Boolean Difference

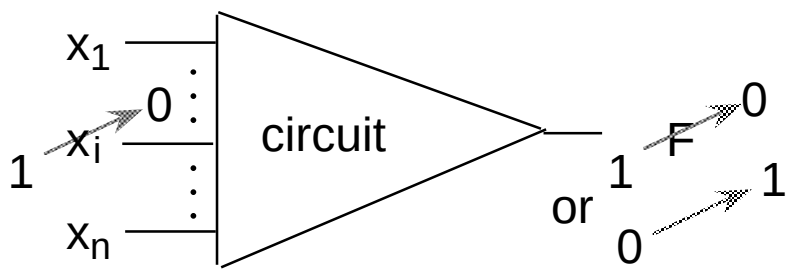
- **Physical Meaning of Boolean Difference**
  - **For a logic function F(X)=F(x_1, ..., x_i, ..., x_n), find all the input combinations that make the change of value in x_i also cause the change of value in F.**
- **Logic Operation of Boolean Difference**
  - **The Boolean difference of F(X) w.r.t.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

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

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

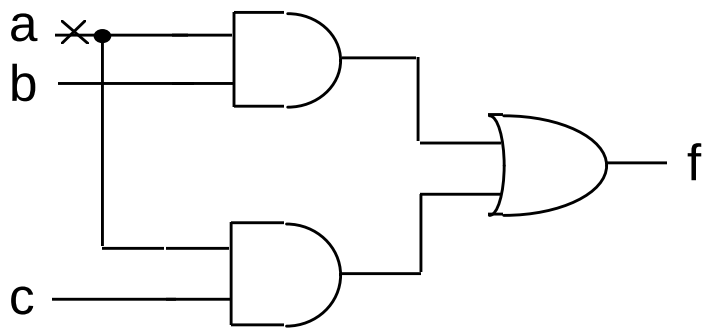
- **Relationship between TG and Boolean Difference**





Applying Boolean Difference to Test Generation (1/2)

Case 1: Faults are present at PIs.

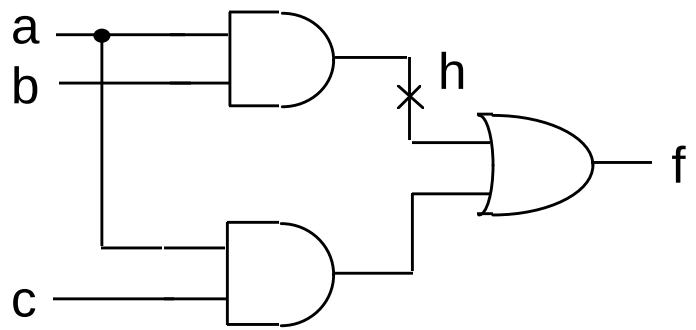


f = ab+ac => $\frac{df}{da} = f_a(0) \oplus f_a(1) = 1 \bullet (b+c)+0 = b+c$

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)}.

Applying Boolean Difference to Test Generation (2/2)

Case 2: Faults are present at internal lines.



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

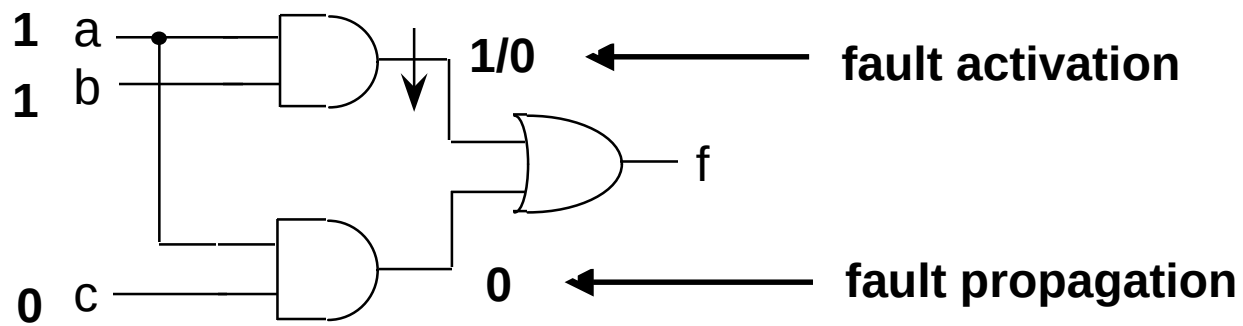
The set of all tests for line *h* s-a-1 is

{ (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)}.

Test Generation Methods
(From Circuit Structure)



- **Two basic goals:**

  Fault activation (FA)

  Fault propagation (FP)

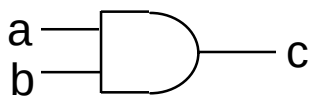
  => Line justification (LJ)

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).

Common Concepts for Structural TG

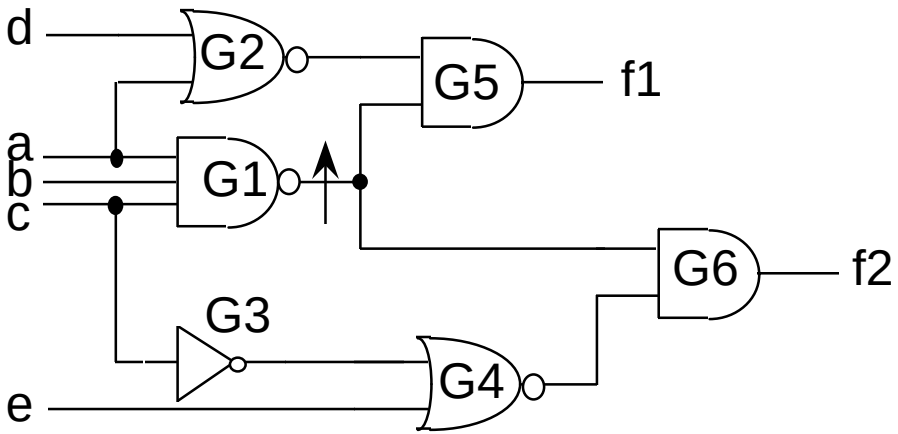
- **The FA problem => a LJ problem.**
- **The FP problem =>**
  (1) Select a FP path to a PO => decisions.
  (2) Once the path is selected => a set of LJ problems.
- **The LJ problems => decisions or implications .**

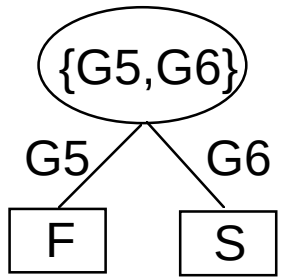
  ex:



  To justify c=1 => a=1 and b=1. (implication)
  To justify c=0 => a=0 or b=0. (need make decisions)
- **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.

Ex: Decisions When Fault Propagation





The corresponding decision tree

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

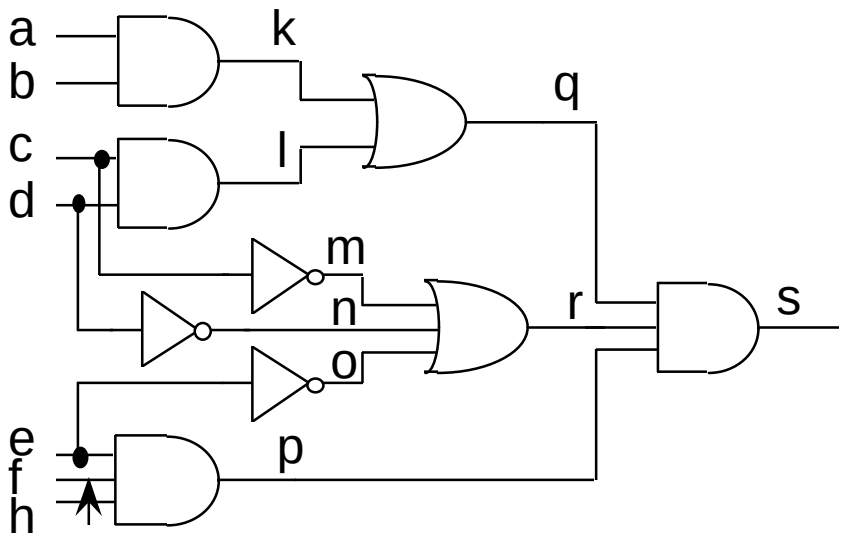
Decision: through G5 => G2=1 => d=0, a=0. => inconsistency => backtracking!!

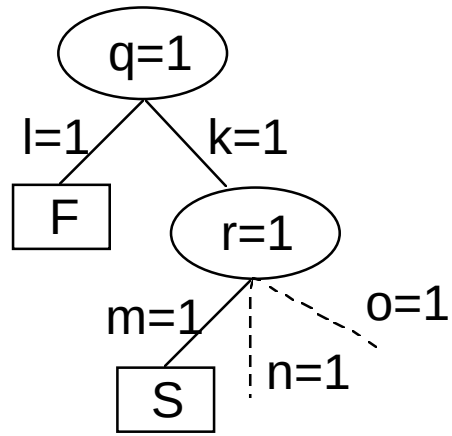
Decision: through G6 => G4=1 => e=0. => done!!

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

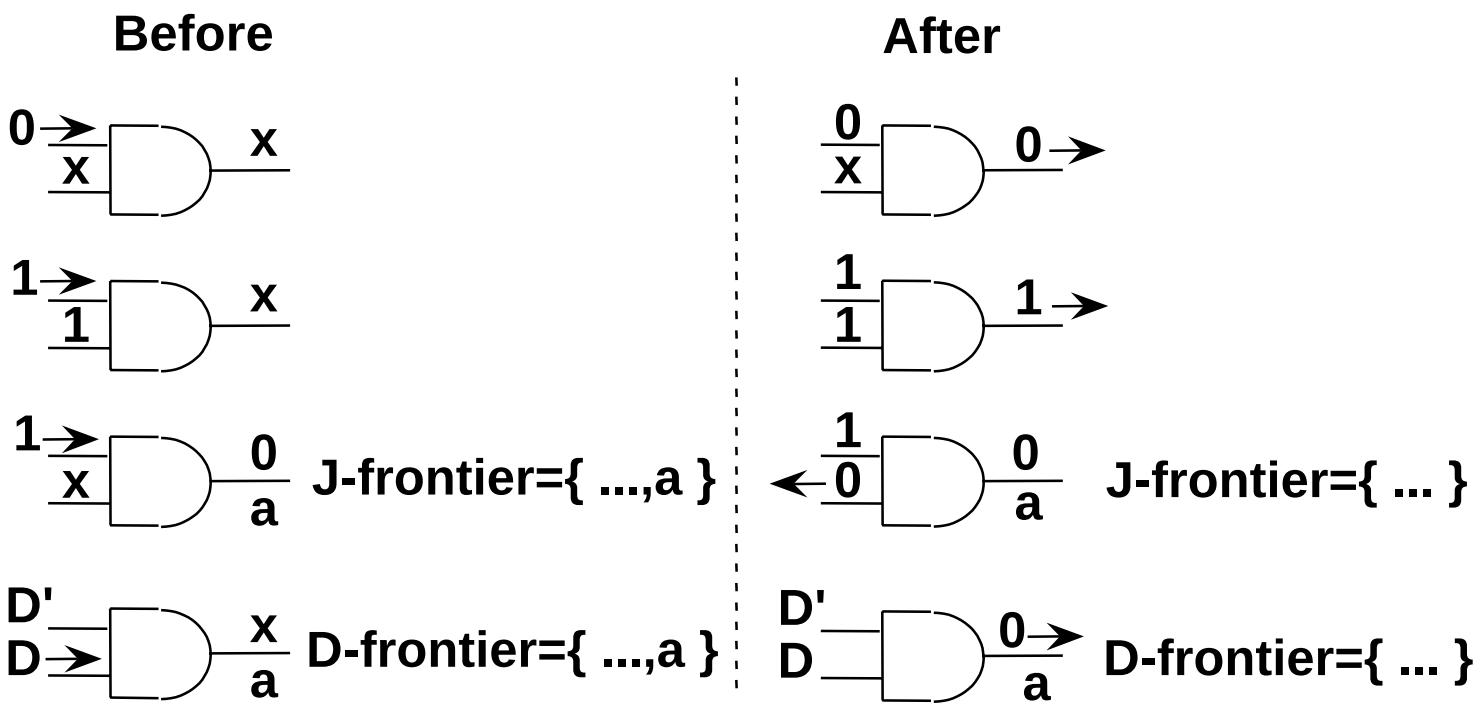
FA => h=D'; FP => e=1, f=1 (=> o=0); FP => q=1, r=1.
To justify q=1 => l=1 or k=1.
Decision: l=1 => c=1, d=1 => m=0, n=0 => r=0. => inconsistency => backtracking!!
Decision: k=1 => a=1, b=1.
To justify r=1 => m=0 or n=0 (=> c=1 or d=1). => done!!

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}.

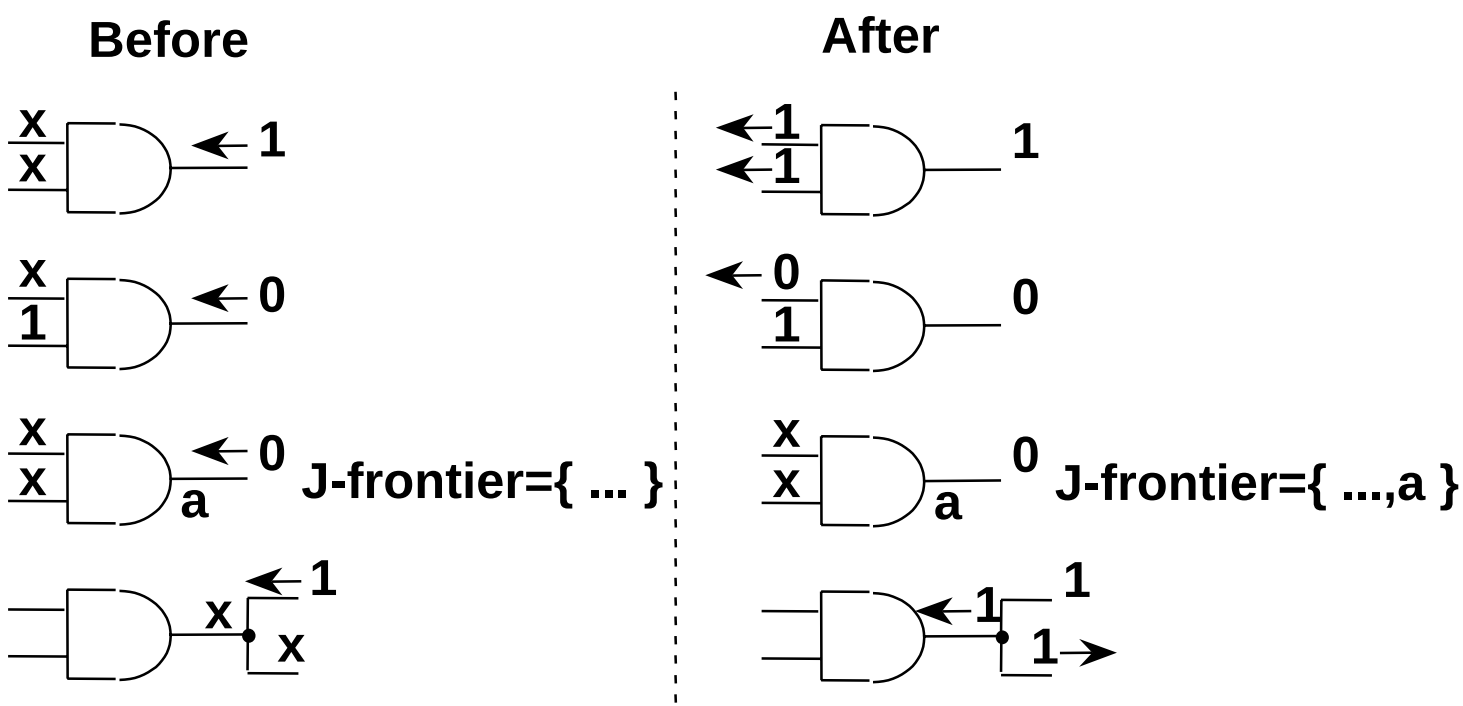
Implications

- **Implication: computation of the values that can be uniquely determined.**
  - **Local implication: propagation of values from one line to its immediate successors or predecessors.**
  - **Global implication: the propagation involving a larger area of the circuit and reconvergent fanout.**
- **Maximum implication principle: perform as many implications as possible.**
- **Maximum implications help us to either reduce the number of problems that need decisions or to reach an inconsistency sooner.**

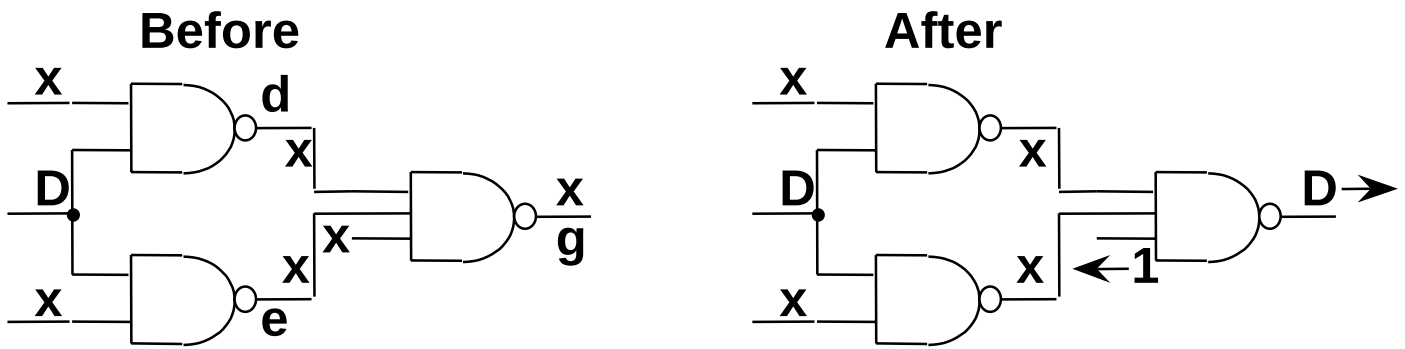
Local Implications (Forward)



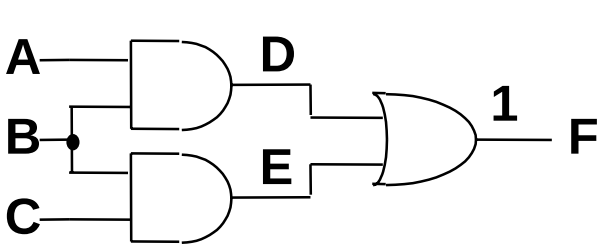
Local Implications (Backward)



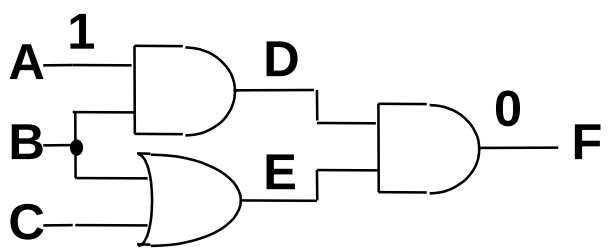
Global Implications



(1) Future unique D-drive.



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

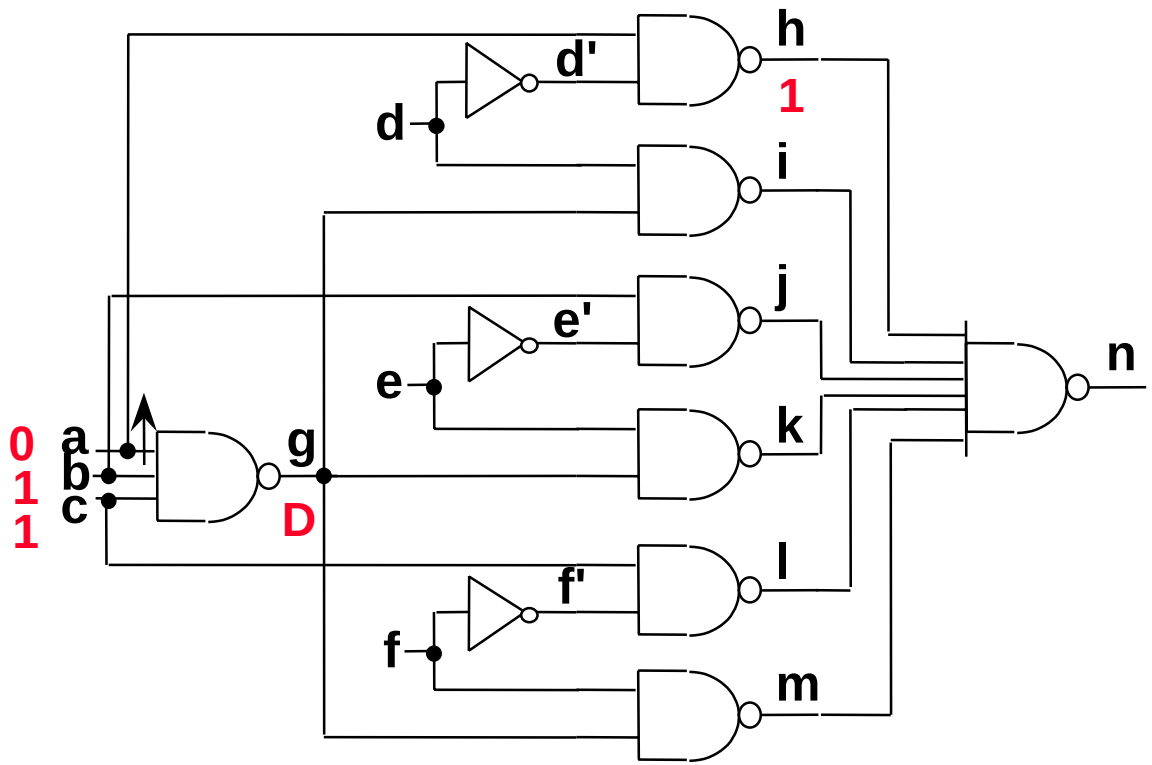


(3) F=0 implies B=0 when A=1.
(Dynamic learning)

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

D-Algorithm: Example

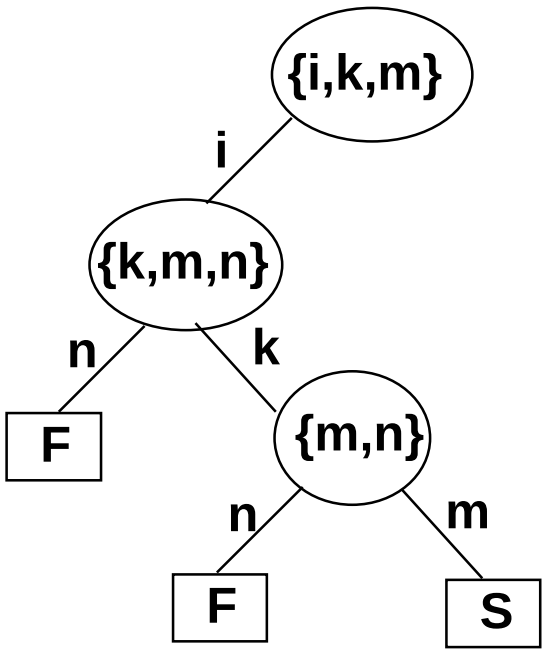
- Logic values = {0, 1, D, D', x}.



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

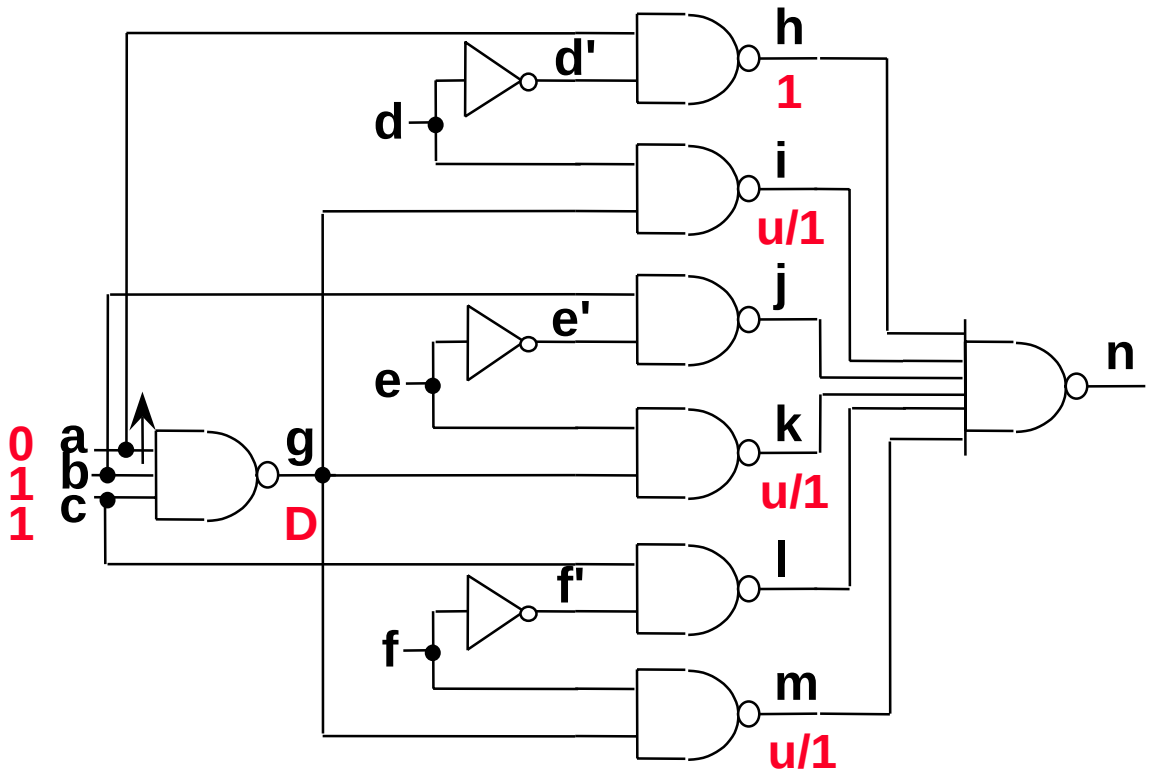


Two times of backtracking!!

- 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.

9-V Algorithm: Example

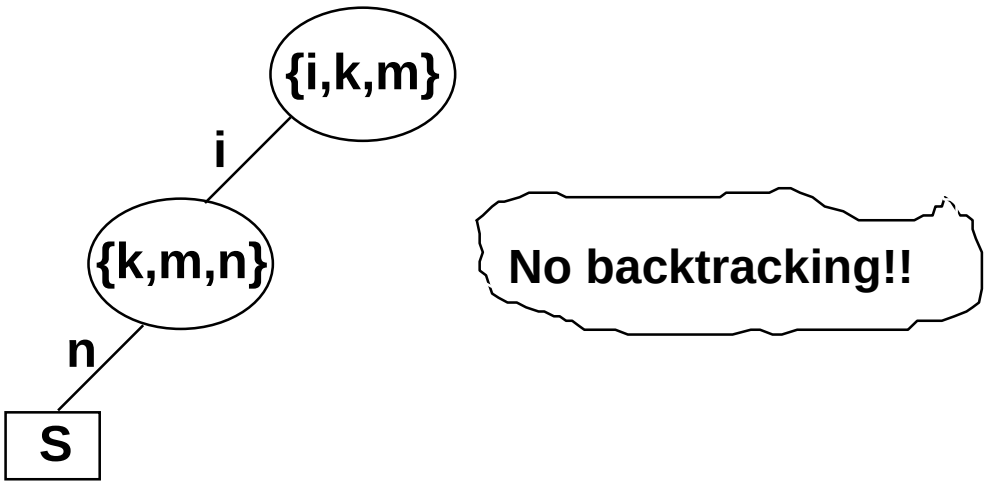
- Logic values = {0/0, 0/1, 0/u, 1/0, 1/1, 1/u, u/0, u/1, u/u}, where 0/u={0,D'}, 1/u={D,1}, u/0={0,D}, u/1={D',1}, u/u={0,1,D,D'}.
- Thus reduces the amount of search done for multiple path sensitization in D-algorithm.



9-V Algorithm: Value Computation

Decision	Implication	Comments
	a=0 h=1 b=1 c=1 g=D i=u/1 k=u/1 m=u/1	Activate the fault Unique D-drive
d=1	 i=D d?0 n=1/u	Propagate via i
l=u/1 j=u/1	 n=D f?u/0 f=1 f?0 e?u/0 e=1 e?0 k=D m=D	Propagate via n

9-V Algorithm: Decision Tree



- The main difference between the D-algorithm and 9-V algorithm is: Whenever there are k possible paths for fault propagation , the D-algorithm may eventually try all the 2^k -1 combinations of paths. However, since the 9-V algorithm tries only one path at a time without precluding simultaneous fault propagation on the other k-1 paths, it will enumerate at most k ways of fault propagation.

PODEM (Path-Oriented Decision Making)

- We have seen that the problems of FA and FP lead to sets of LJ problems. The LJ problems can be solved via value assignments.
- In D-algorithm, the value assignments are allowed on any internal lines. => backtracking could occur at any line.
- However, PODEM allows value assignments only on PIs. => backtracking can occur only at the PIs.
  - It treats a value V_k to be justified for line k as an objective (k,V_k).
  - A backtracing procedure maps the objective into a PI assignment that is likely to contribute to achieving the objective.
  - Why called PODEM (Path-Oriented DEcision Making) ?

A Simple Backtracing Procedure

Objective = (k,V_k).

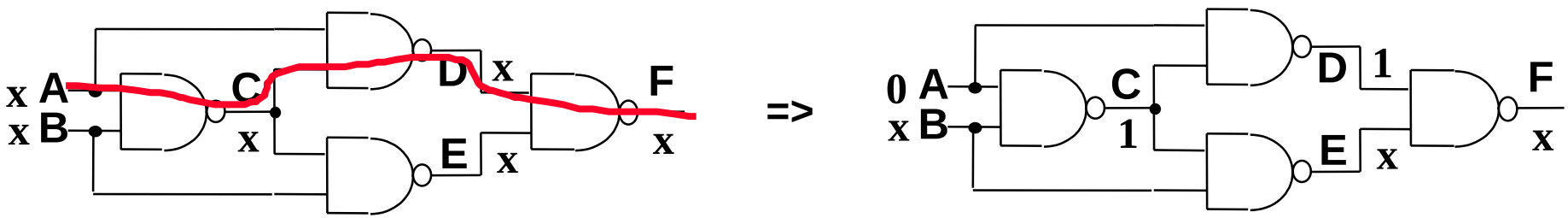
Step 1. Find a x-path from line k to a PI, say A.

Step 2. Count the inversion parity of the path.

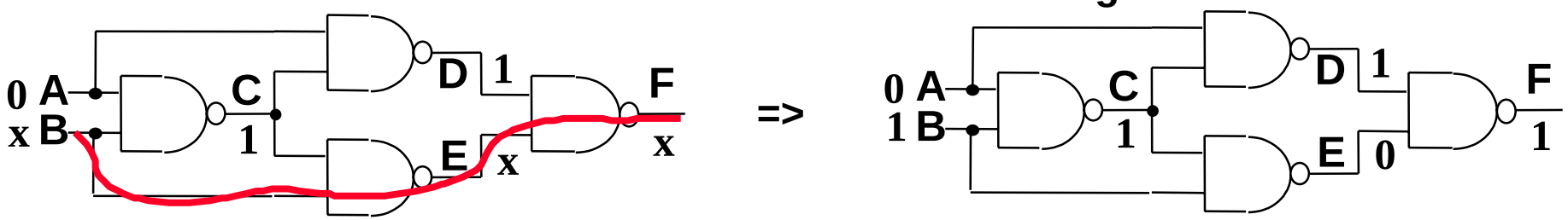
Step 3. If the inversion parity is even => return (A, V_k). Otherwise => return (A, V_k').

**** A path is a x-path if all of its lines have value x.***

Ex: Objective = (F,1).

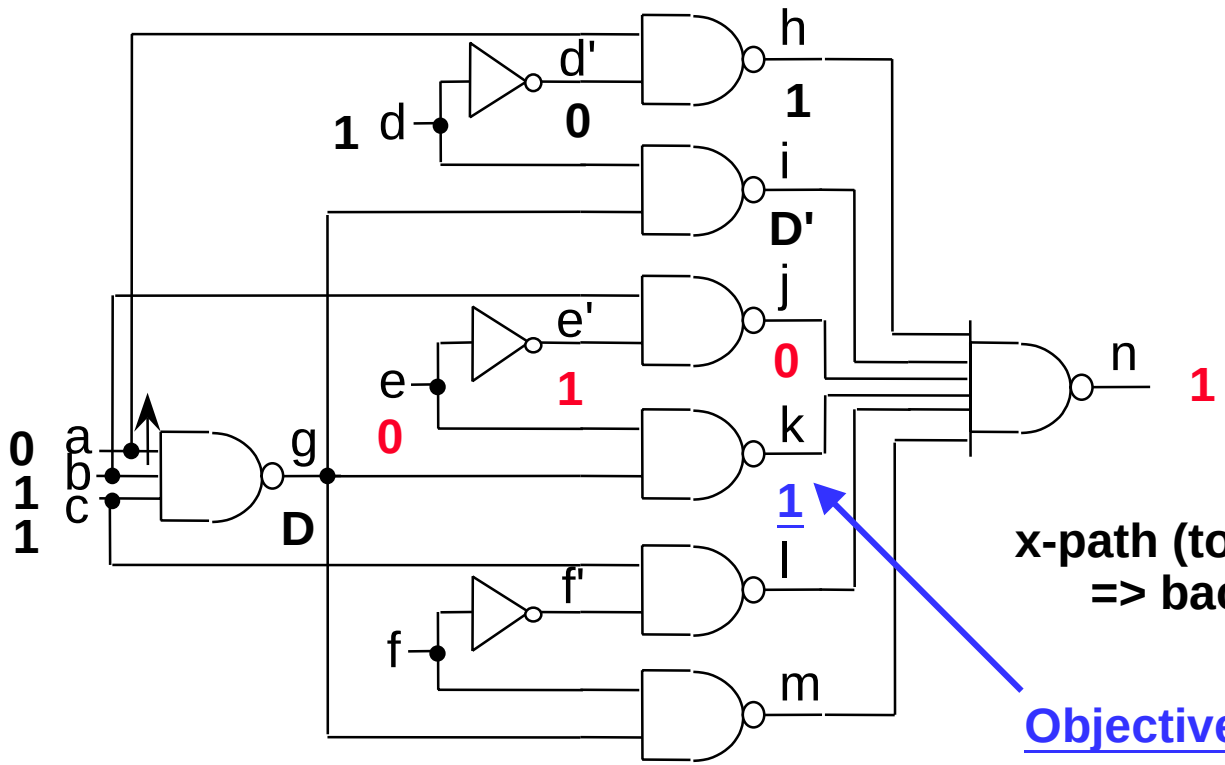


The first time of backtracing



The second time of backtracing

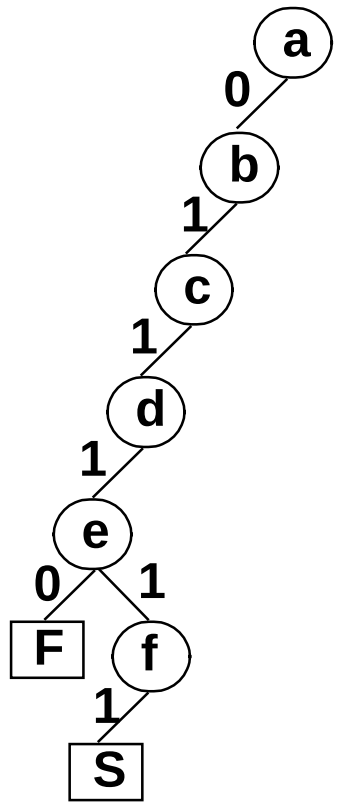
PODEM: Example 1



PODEM: Value Computation

Objective	PI assignment	Implications	D-frontier	Comments
a=0	a=0	h=1	g	
b=1	b=1		g	
c=1	c=1	g=D	i,k,m	
d=1	d=1	d?0 i=D	 k,m,n	
k=1	e=0	e?1 j=0 k=1 n=1	 m	 x-path check fail !!
	e=1	e?0 j=1 k=D	 m,n	reversal
l=1	f=1	f?0 l=1 m=D n=D		

PODEM: Decision Tree



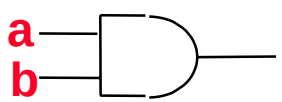
*** decision node: the PI selected to assign value.**
branch: the value assigned to the PI.

A More Intelligent Backtracing

- **To guide the backtracing process of PODEM, controllability for each line is measured.**
  - **CY1(a): the probability that line a has value 1.**
  - **CY0(a): the probability that line a has value 0.**

  Ex: f = ab. Assume CY1(a)=CY0(a)=CY1(b)=CY0(b)=0.5.
  => CY1(f)=CY1(a)xCY1(b)=0.25, CY0(f)=CY0(a)+CY0(b)-CY0(a)xCY0(b)=0.75.
- **How to guide the backtracing using controllability?**
  - **Principle 1: Among several unsolved problems, first attack the hardest one.**
  - **Principle 2: Among several solutions of a problem, first try the easiest one.**

  ex:

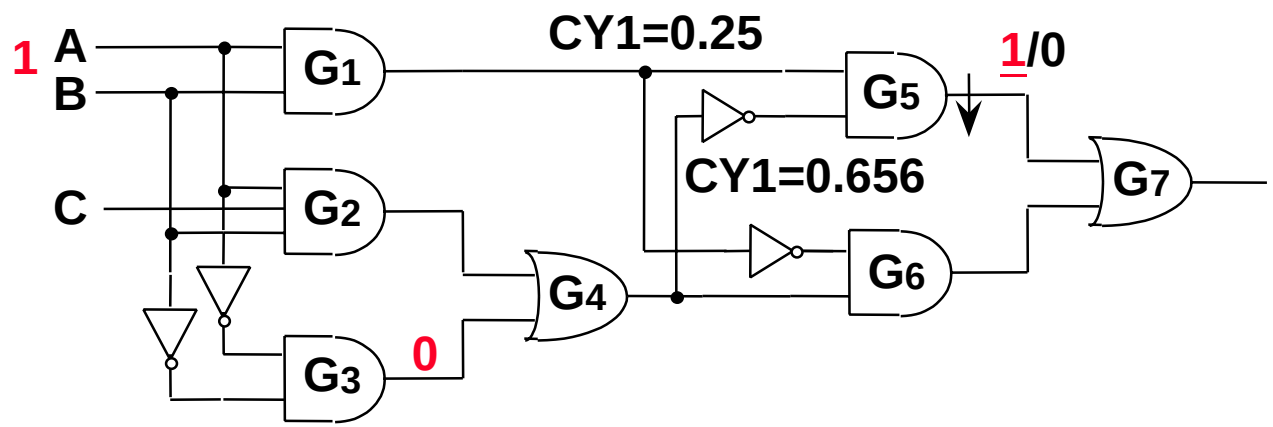


CY1(a) = 0.33, CY1(b) = 0.5
CY0(a) = 0.67, CY0(b) = 0.5

Objective=(c,1) => Choose *path c-a* to backtracing.
Objective=(c,0) => Choose *path c-a* to backtracing.

PODEM: Example 2 (1/3)

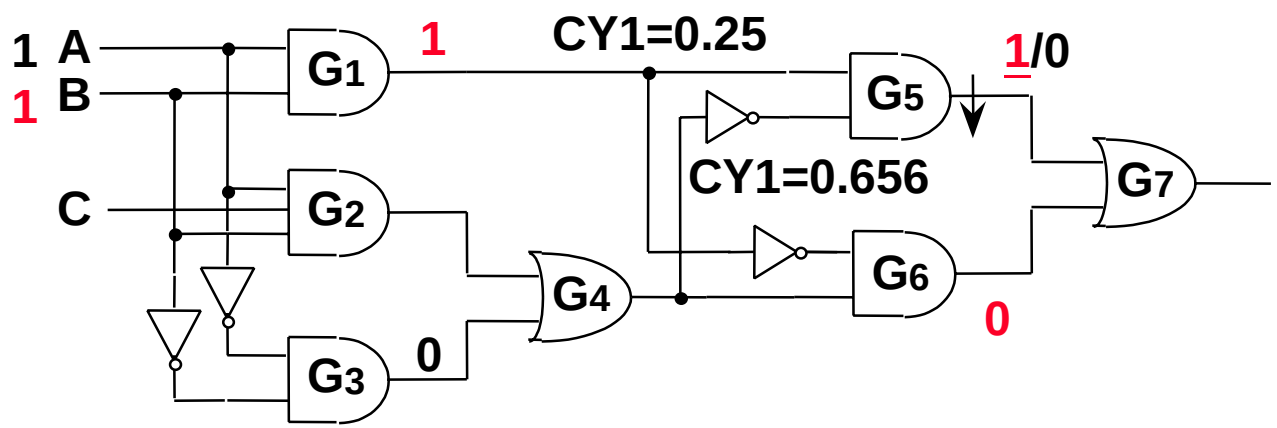
Initial objective=(G5,1).
G5 is an AND gate => Choose the hardest-1 => Current objective=(G1,1).
G1 is an AND gate => Choose the hardest-1 => Arbitrarily, Current objective=(A,1).
A is a PI => Implication => G3=0.



The controllabilities are calculated by a testability measure program TEA.
Initially, CY1 and CY0 for all PIs are set to 0.5.

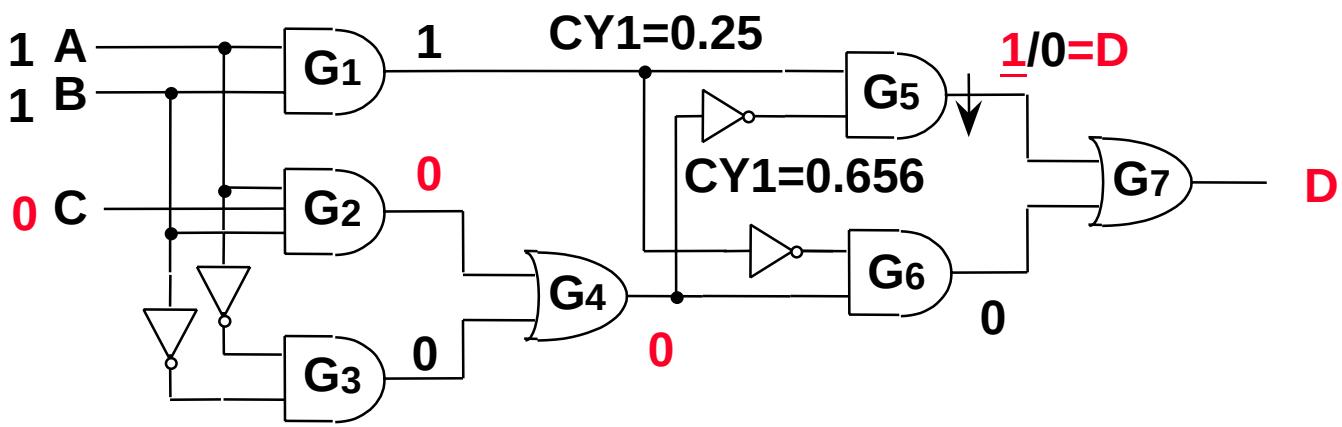
PODEM: Example 2 (2/3)

The initial objective satisfied? No! => Current objective=(G5,1).
G5 is an AND gate => Choose the hardest-1 => Current objective=(G1,1).
G1 is an AND gate => Choose the hardest-1 => Arbitrarily, Current objective=(B,1).
B is a PI => Implication => G1=1, G6=0.



PODEM: Example 2 (3/3)

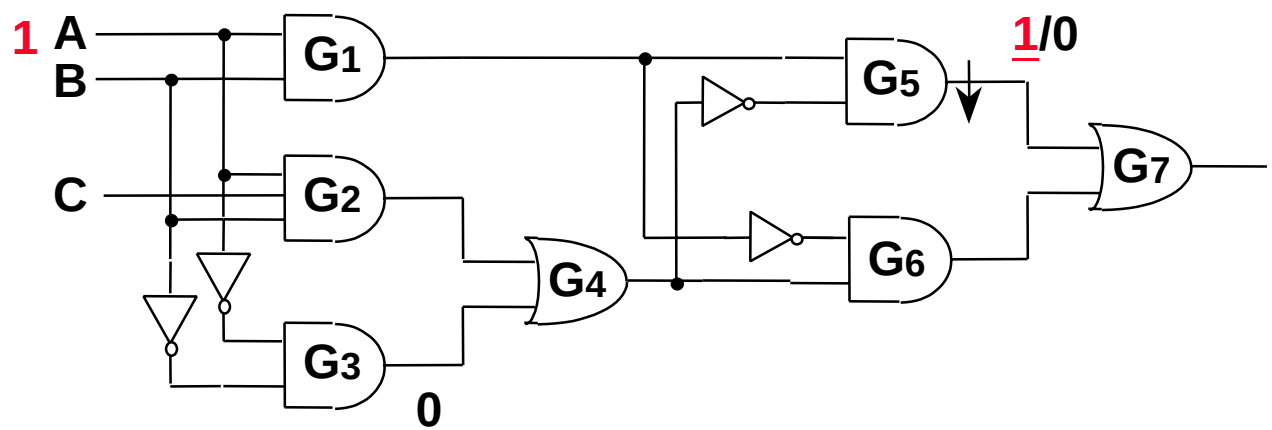
The initial objective satisfied? No! => Current objective=(G5,1).
The value of G1 is known => Current objective=(G4,0).
The value of G3 is known => Current objective=(G2,0).
A, B is known => Current objective=(C,0).
C is a PI => Implication => G2=0, G4=0, G5=D, G7=D.



No backtracking!!

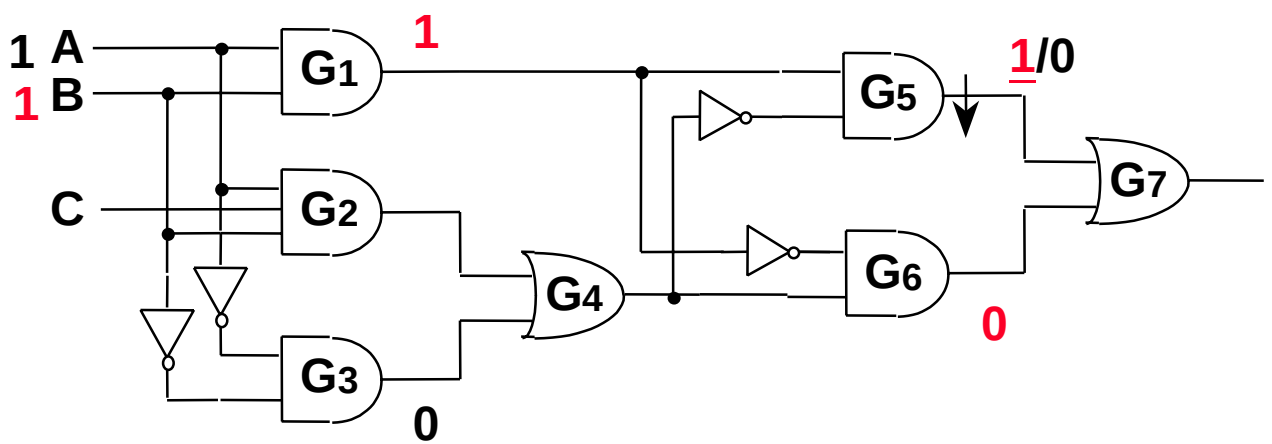
If The Backtracing Is Not Guided (1/3)

Initial objective=(G5,1).
Choose path G5-G4-G2-A => A=0.
Implication for A=0 => G1=0, G5=0 => Backtracking to A=1.
Implication for A=1 => G3=0.



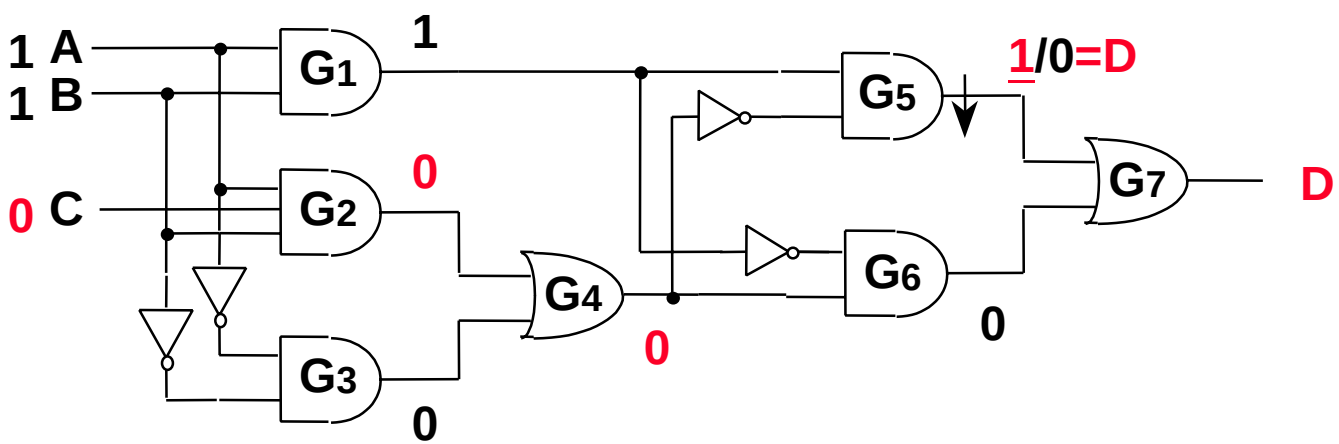
If The Backtracing Is Not Guided (2/3)

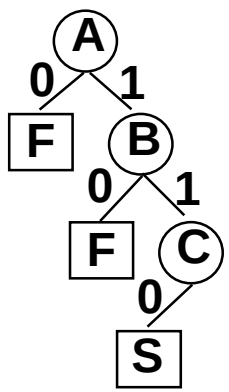
The initial objective satisfied? No! => Current objective=(G5,1).
Choose path G5-G4-G2-B => B=0.
Implication for B=0 => G1=0, G5=0 => Backtracking to B=1.
Implication for B=1 => G1=1, G6=0.



If The Backtracing Is Not Guided (3/3)

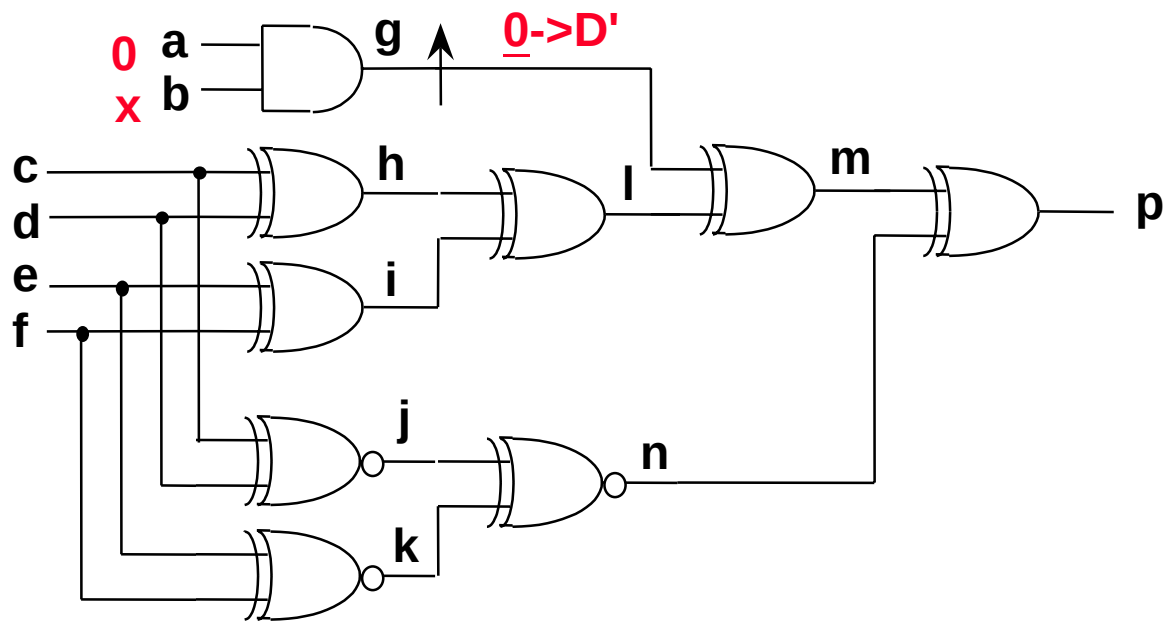
The initial objective satisfied? No! => Current objective=(G5,1).
Choose path G5-G4-G2-C => C=0.
Implication for C=0 => G2=0, G4=0, G5=D, G7=D.





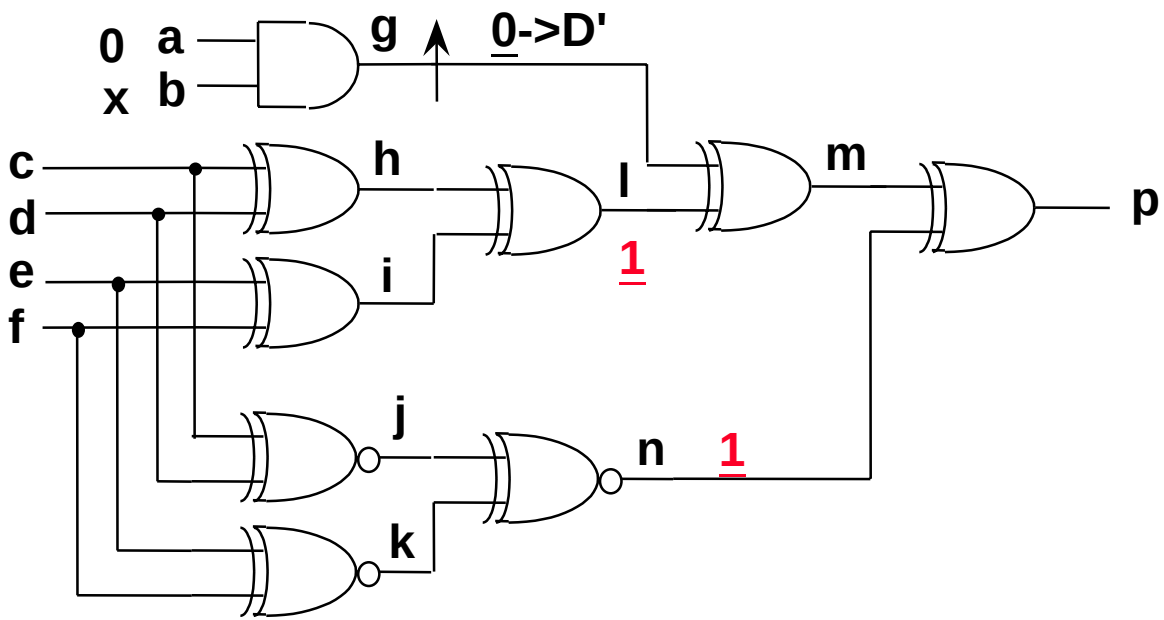
Two times of backtracking!!

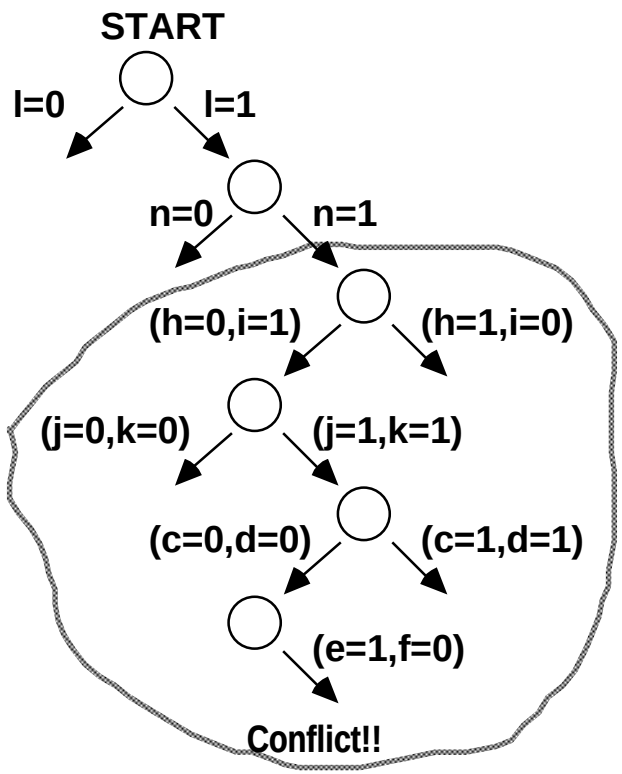
ECAT Circuit: PODEM



No backtracking !!

ECAT Circuit: D-Algorithm



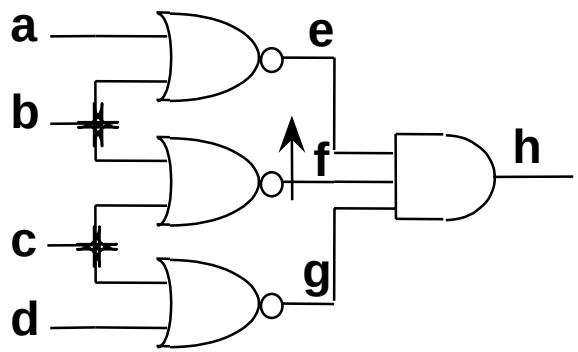


Features of PODEM

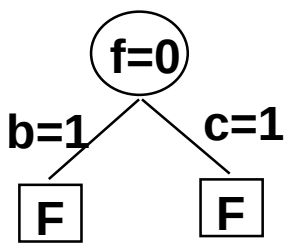
- **PODEM examines all possible input patterns implicitly but exhaustively (branch-and-bound) as tests for a given fault. => It is a complete TG.**
- **PODEM does not need**
  - **consistency check, as conflicts can never occur;**
  - **the J-frontier, since ther are no values that require justification;**
  - **backward implication, because values are propagated only forward.**
- **Backtracking is implicitly done by simulation rather than by an explicitly save/restore process. (State saving and restoring is a time-consuming process.)**
- **Experimental results show that PODEM is generally faster than the D-algorithm. [4]**

Redundant Faults

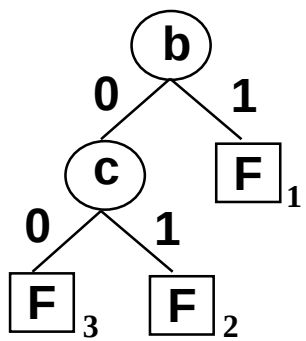
- Presence of the fault does not change the functionality of the circuit under test.
- Ex:



- Good function: h = (a+b)' (b+c)' (c+d)' = a'b'c'd'
  Faulty function = (a+b)' (c+d)' = a'b'c'd'
- Perform test generation



decision tree for D-algorithm

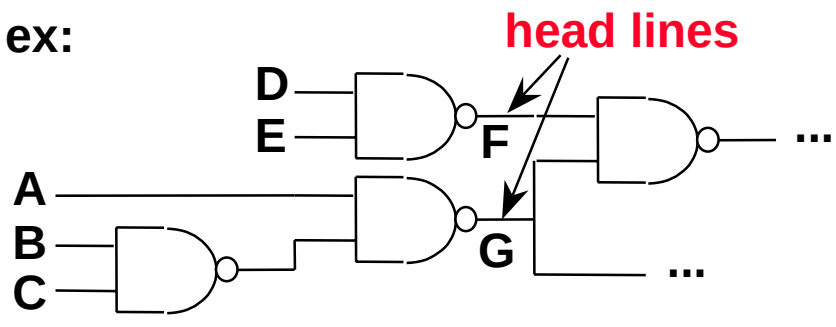


decision tree for PODEM

FAN (Fanout-Oriented TG)

- **FAN introduces two major extensions to the backtracing concept of PODEM:**
  1. **Rather than stopping at PIs, backtracing in FAN may stop at internal lines. => will reduce the number of backtracking.**

ex:



Backtracing stops at lines F and G rather than A,B,C,D,and E.

  2. **Rather than trying to satisfy one objective, FAN uses a multiple-backtrace procedure that attempts to simultaneously satisfy a set of objectives. (In PODEM, a PI assignment satisfying one objective may preclude achieving another one, and this leads to backtracking.)**

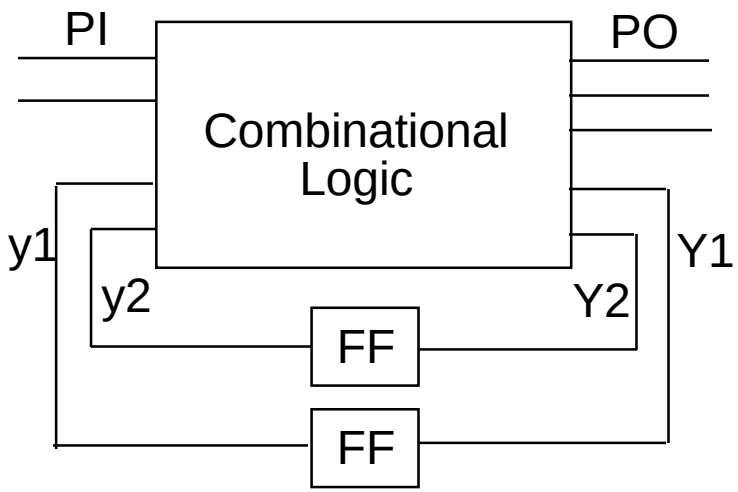
ATPG

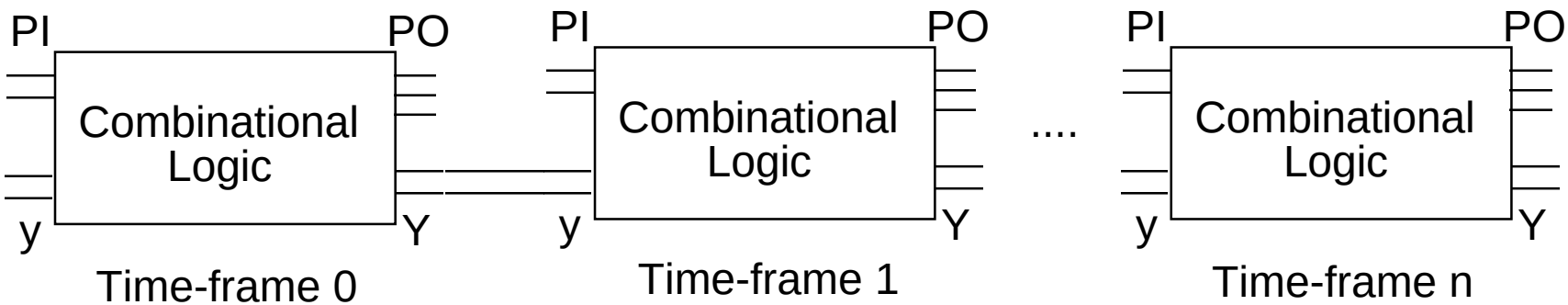
- **ATPG (Automatic Test Pattern Generation):**
  - **Generate a set of test patterns for a set of target faults.**
- **Basic scheme:**
  - **Initialize vector set to NULL**
  - **Repeat**
    - **Generate a new test vector**
    - **Evaluate fault coverage for the test vector**
    - **If the test vector is acceptable Then add it to vector set**
  - **Until required fault coverage is obtained**
- **To accelerate the ATPG:**
  - **Random patterns are often generated first to detect easy- to-detect faults, then a deterministic TG is performed to generate tests for the remaining faults.**

Sequential Test Generation

- **For Circuits with Unknown Initial States**
  - **Time-frame Expansion Based: Extended D-algorithm (IEEE TC, 1971), 9-V Algorithm (IEEE TC, 1976), EBT (DAC, 1978 & 1986), BACK (ICCD, 1988), ...**
  - **Simulation-Based: CONTEST (IEEE TCAD, 1989), TVSET (FTCS, 1988), ...**
- **For Circuits with Known Initial States**
  - **STALLION (IEEE TCAD, 1988), STEED (IEEE TCAD, May 1991), ...**

Iterative Logic Array (ILA) Model for Sequential Circuits



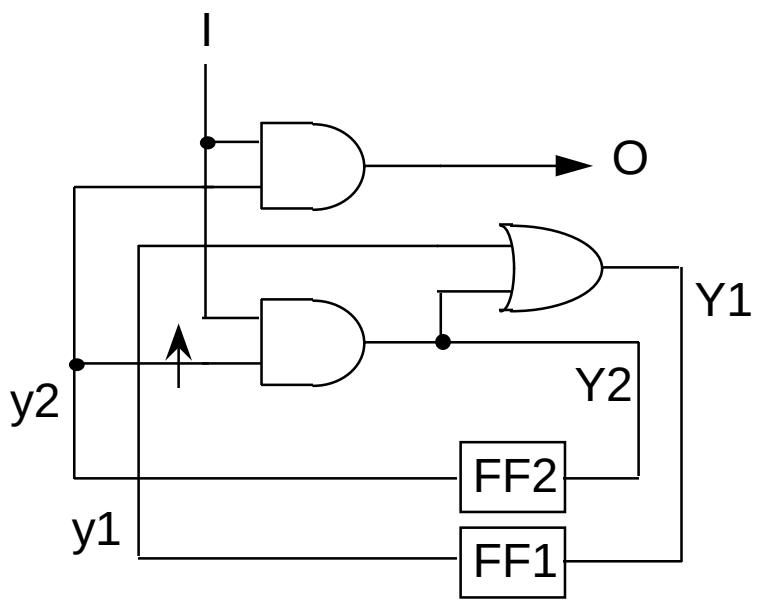


Extended D-Algorithm [1]

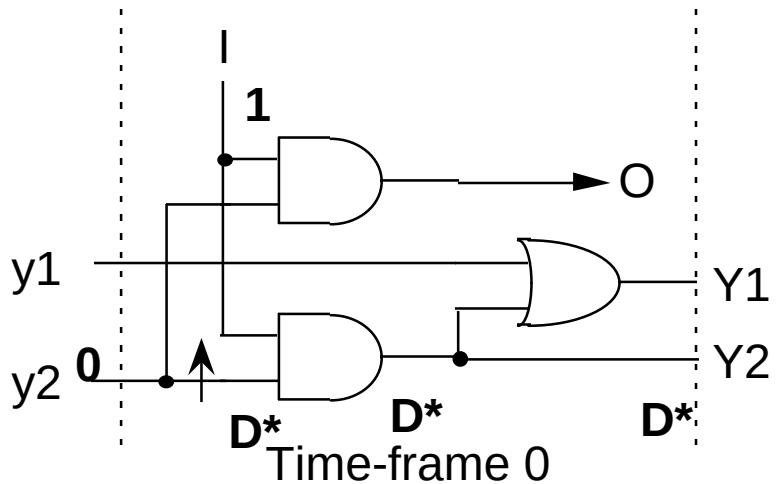
(Kubo, NEC Research & Development, Oct. 1968)
(Putzolu and Roth, IEEE TC, June 1971)

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.**

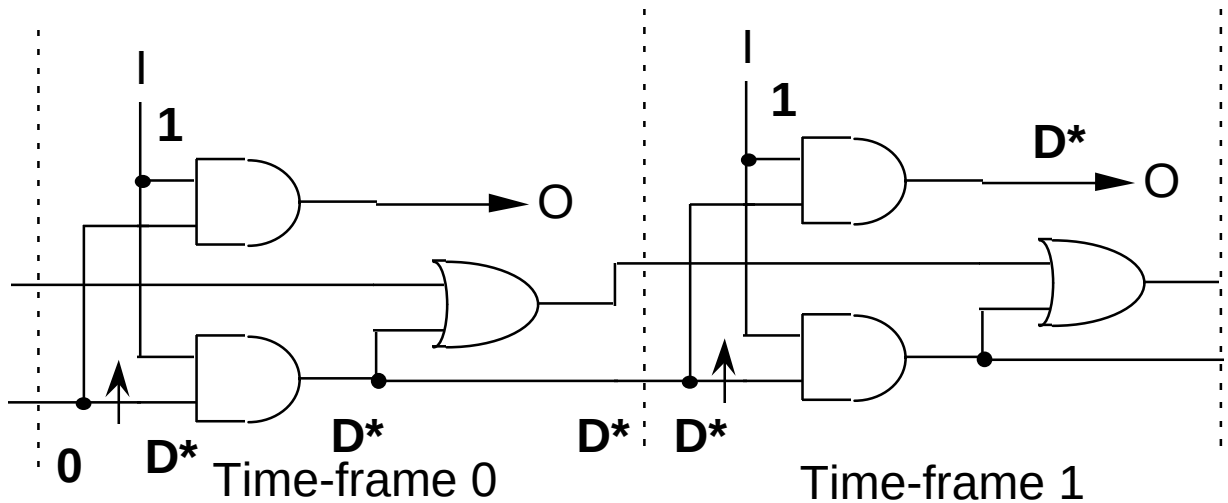
Example for Extended D-Algorithm



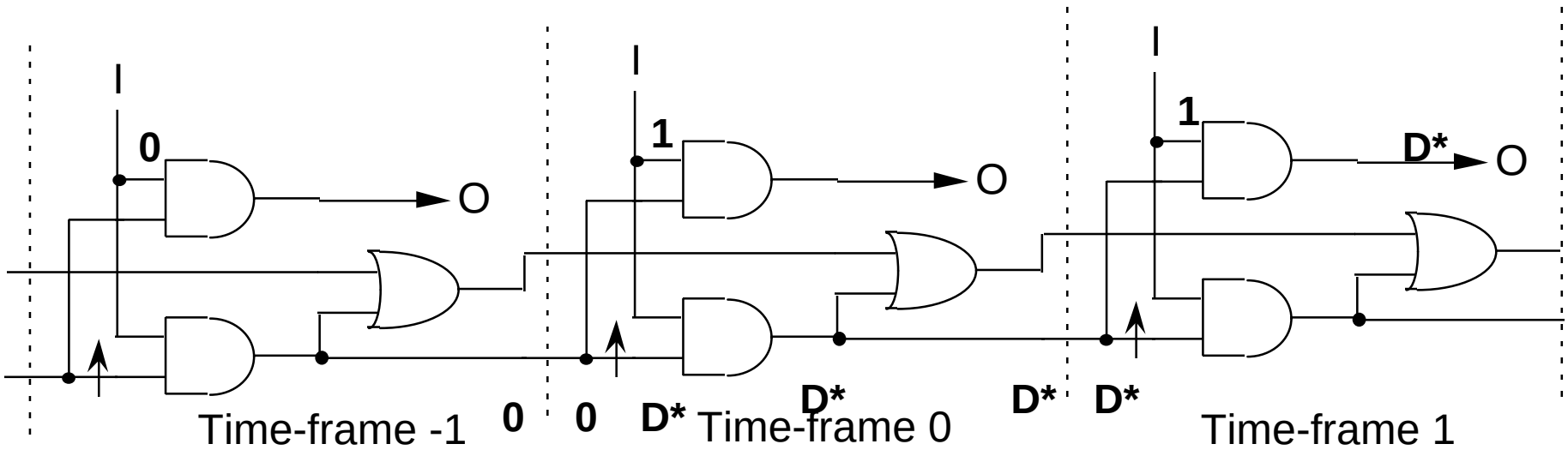
Example: Step 1



Example: Step 2



Example: Step 3

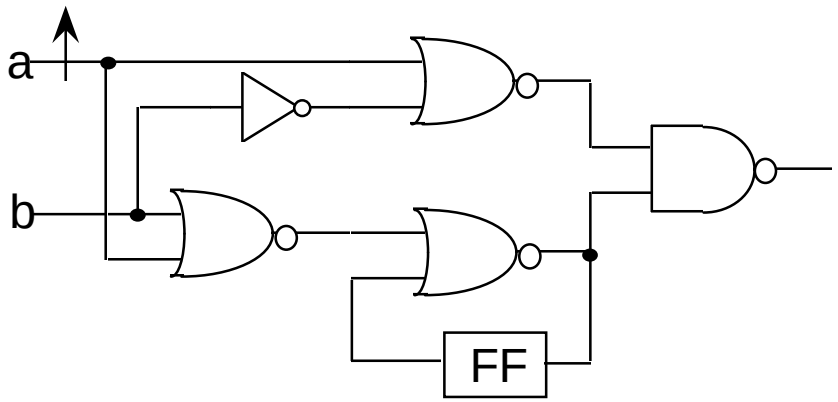


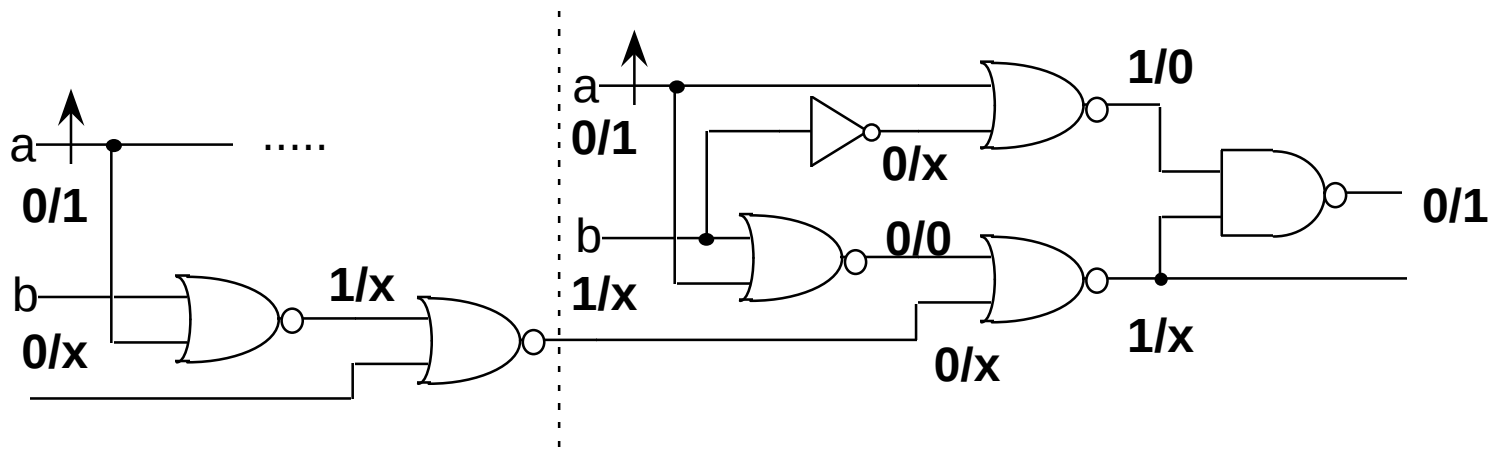
9-V Sequential TG [2]

(Muth, IEEE TC, June 1976)

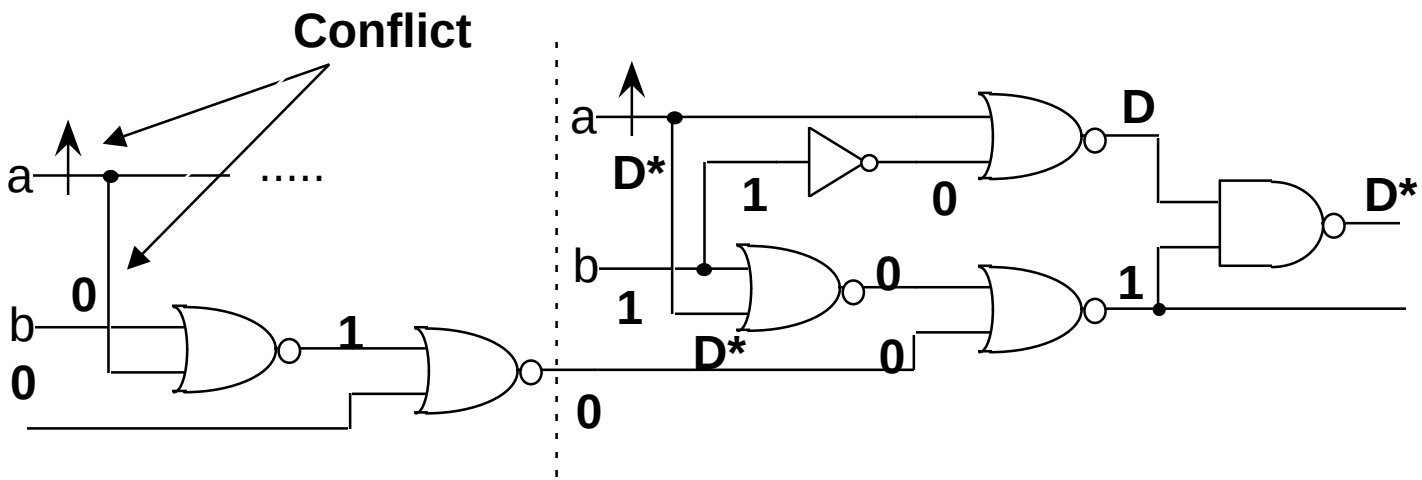
- **Extended D-algorithm is not complete.**
- **If nine-value, instead of five-value, is used, it will be a complete algorithm. (Since it takes into account the possible repeated effects of the fault in the iterative array model.)**

Example: Nine-Valued TG





If Five-Valued TG Is Used



The test can not be generated by five-value TG.

Problems of Mixed Forward and Reverse Time Processing Approaches

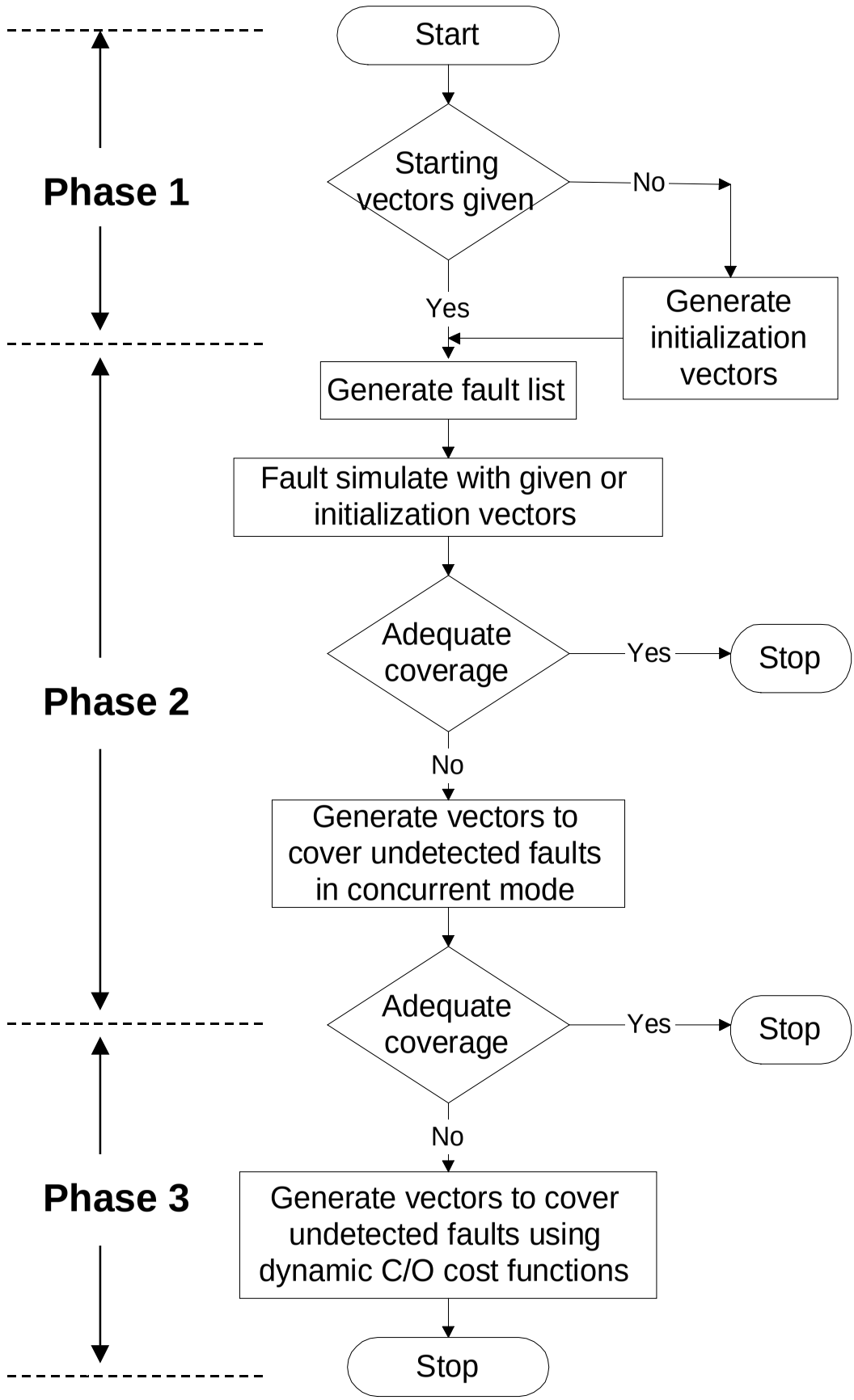
- **The requirements created during the forward process (fault propagation) have to be justified by the backward process later.**
  - **Need going both forward and backward time frames.**
  - **May need to maintain a large number of time-frames during test generation.**
  - **Hard to identify "cycles" .**
  - **Implementation is complicated.**

CONTEST: A Concurrent Test Generator for Sequential Circuits [3,4]

(Agrawal and Cheng, IEEE TCAD, Feb. 1989)

- **Simulated-based test generation.**
- **It is subdivided into three phases :**
  - **Initialization**
  - **Concurrent fault detection**
  - **Single Fault detection**
- **For different phases, different cost functions are defined to guide the searching for vectors.**

Flow Chart of CONTEST



Simulation-Based Approaches

Advantages:

- **Timing is considered.**
- **Asynchronous circuits can be handled.**
- **Can be easily implemented by modifying a fault simulator.**

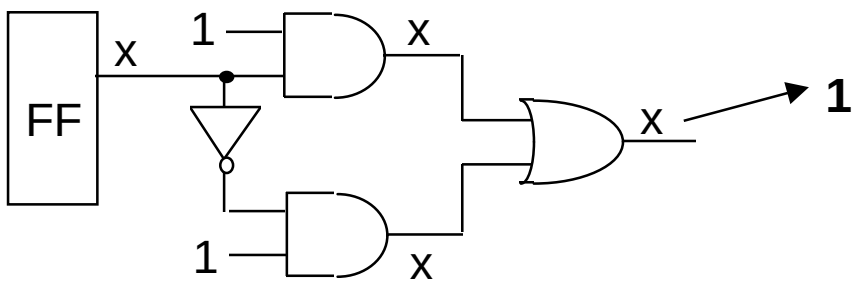
Disadvantages:

- **Can not identify undetectable faults.**
- **Hard-to-activate faults may not be detected.**

Difficulties of Sequential Test Generation

- **Initialization is difficult.**
  - **Justifying invalid states**
  - **Long initialization sequence**
  - **Simulator limitations**

ex:



- **Timing can not be considered by time-frame expansion.**
  - **Generated tests may cause races and hazards.**
  - **Asynchronous circuits can not be handled.**

Test Generation Assuming A known Initial State

- **Initialization is avoided.**
- **Assumption is valid for pure controllers that usually have a reset mechanism (reset PI, resetable flip-flops, ...).**

STALLION [5]

(Ma et al, IEEE TCAD, Oct. 1988)

1. Generate state transition graph (STG) for fault-free circuit.
2. Create a copy of the combinational logic, set it time-frame 0. Generate a test for the fault using PODEM for the time-frame.
3. When the fault is propagated to PPOs (but not POs), find a fault propagation sequence T to propagate the fault effect to a PO using the STG.
4. When there are values required in PPIs, say state S, find a transfer sequence T0 from initial state S0 to S using the STG.
5. Fault simulate sequence T0+T. If it is not a valid test, go to 3 to find another sequence.

STALLION

Advantages:

- **Transfer sequences are easily derived from STG.**
- **Good performance for controllers whose STG can be extracted easily.**

Disadvantages:

- **Fault-free transfer sequence may not be valid.**
- **Extraction of STG may not be feasible for large circuits.**

Heuristics:

- **Construct partial STG only.**
- **If the required transfer sequence can not be derived from partial STG, augment the partial STG.**

References

[1] G. R. Putzolu and T. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits", IEEE Trans. Computers, pp. 639-647, June 1971.

[2] P. Muth, "A Nine-Valued Circuit Model for Test Generation", IEEE Trans. Computers, pp. 630-636, June 1976.

[3] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A Directed Search Method for Test Generation Using a Concurrent Simulator", IEEE Trans. CAD, pp. 131-138, Feb. 1989.

[4] K. T. Cheng and V. D. Agrawal, "Unified Methods for VLSI Simulation and Test Generation", Chapter 7, Kluwer Academic Publishers, 1989.

[5] H-K. T. Ma, et al, "Test Generation for Sequential Circuits", IEEE Trans. CAD, pp. 1081-1093, Oct. 1988.