Automatic Test Pattern Generation - II

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Dept. of Electrical Engineering
Indian Institute of Technology Bombay
viren@ee.iitb.ac.in

EE 709: Testing & Verification of VLSI Circuits

Lecture – 12 (Jan 30, 2012)

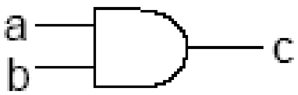
ATPG - Algorithmic

- Path Sensitization Method
  - Fault Sensitization
  - Fault Propagation
  - Line Justification

- Path Sensitization Algorithms
  - D- Algorithm (Roth)
  - PODEM (P. Goel)
  - FAN (Fujiwara)
  - SOCRATES (Schultz)
  - SPIRIT (Emil & Fujiwara)

Common Concept

- Fault Activation problem → a LJ Problem
- The Fault Propagation problem →
  1. Select a FP path to PO → Decision
  2. Once the path is selected → a set of LJ problems
- The LJ Problems → Decisions or Implications



To justify c = 1 → a = 1, b = 1 (Implication)

To justify c = 0 → a = 0 or b = 0 (Decision)

- Incorrect decision → Backtrack → Another decision

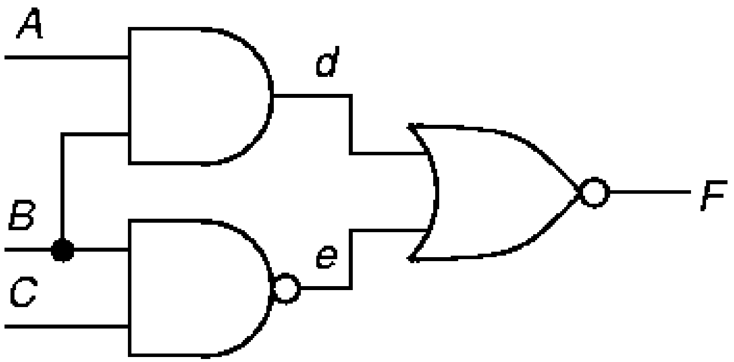
D-Algorithm

Roth (IBM) - 1966

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - D-Calculus (5 valued logic)
  - Implications – forward and backward
  - Implication stack
  - Backtrack
  - Test Search Space

Singular Cover Example

- **Minimal set of logic signal assignments to show *essential prime implicants* of *Karnaugh map***



Gate	Inputs		Output	Gate	Inputs		Output
AND	*A*	*B*	*d*	NOR	*d*	*e*	*F*
1	0	X	0	1	1	X	0
2	X	0	0	2	X	1	0
3	1	1	1	3	0	0	1

D-Cube

- **Collapsed truth table entry to characterize logic**
- **Use Roth's 5-valued algebra**
- **Can change all D's to $\overline{D}$'s and $\overline{D}$'s to D's (do both)**
- **AND gate:**

	A	*B*	*d*
Rows 1 & 3 Reverse inputs And two cubes Interchange D and $\overline{D}$	D 1 D $\overline{D}$ 1 $\overline{D}$	1 D D $\overline{D}$ $\overline{D}$ 1	D D D $\overline{D}$ $\overline{D}$ $\overline{D}$

D-Cube Operation of D-Intersection

- ψ – undefined (same as ϕ)
- μ or λ – requires inversion of **D** and **$\bar{D}$**
- ***D-intersection*: $0 \cap 0 = 0 \cap X = X \cap 0 = 0$**

  $1 \cap 1 = 1 \cap X = X \cap 1 = 1$

  $X \cap X = X$

$\cap$	0	1	X	D	$\bar{D}$
0	0	ϕ	0	ψ	ψ
1	ϕ	1	1	ψ	ψ
X	0	1	X	D	$\bar{D}$
D	ψ	ψ	D	μ	λ
$\bar{D}$	ψ	ψ	$\bar{D}$	λ	μ

- ***D-containment* –**
  Cube *a* contains Cube *b* if *b* is a subset of *a*

Primitive D-Cube of Failure

- **Models circuit faults:**
  - ***Stuck-at-0***
  - ***Stuck-at-1***
  - ***Bridging fault* (short circuit)**
  - **Arbitrary change in logic function**
- **AND Output sa0: “1 1 D”**
- **AND Output sa1: “0 X $\overline{D}$ ”**
  “X 0 $\overline{D}$ ”
- **Wire sa0: “D”**
- ***Propagation D-cube* – models conditions under which fault effect propagates through gate**

Implication Procedure

1. **Model fault with appropriate *primitive D-cube of failure* (PDF)**
2. **Select *propagation D-cubes* to propagate fault effect to a circuit output (*D-drive* procedure)**
3. **Select *singular cover* cubes to justify internal circuit signals (*Consistency* procedure)**

- **Put signal assignments in *test cube***
- **Regrettably, cubes are selected very arbitrarily by D-ALG**

D-Algorithm – Top Level

1. **Number all circuit lines in increasing level order from PIs to POs;**
2. **Select a primitive D-cube of the fault to be the *test cube*;**
   - **Put logic outputs with inputs labeled as D ($\overline{D}$) onto the *D-frontier*;**
3. ***D-drive* ();**
4. ***Consistency* ();**
5. **return ();**

D-Algorithm – *D-drive*

while (untried fault effects on D-frontier)

select next untried D-frontier gate for propagation;

while (untried fault effect fanouts exist)

select next untried fault effect fanout;

generate next untried propagation D-cube;

D-intersect selected cube with test cube;

if (intersection fails or is undefined) continue;

if (all propagation D-cubes tried & failed) break;

if (intersection succeeded)

add propagation D-cube to test cube -- recreate *D-frontier*;

Find all forward & backward implications of assignment;

save *D-frontier*, algorithm state, test cube, fanouts, fault;

break;

else if (intersection fails & D and $\overline{D}$ in test cube) *Backtrack* ();

else if (intersection fails) break;

if (all fault effects unpropagatable) *Backtrack* ();

D-Algorithm - *Consistency*

g = coordinates of test cube with 1's & 0's;
if (*g* is only PIs) fault testable & stop;
for (each unjustified signal in *g*)
    Select highest # unjustified signal *z* in *g,* not a PI;
    if (inputs to gate *z* are both D and $\overline{D}$) break;
    while (untried singular covers of gate *z*)
        select next untried singular cover;
        if (no more singular covers)
            If (no more stack choices) fault untestable & stop;
            else if (untried alternatives in *Consistency*)
                pop implication stack -- try alternate assignment;
            else
                Backtrack ();
                D-drive ();
        If (singular cover D-intersects with *z*) delete *z* from *g*, add inputs to singular cover to *g, find all forward and backward implications of new assignment,* and break;
        If (intersection fails) mark singular cover as failed;

Backtrack

if (PO exists with fault effect) *Consistency* ();

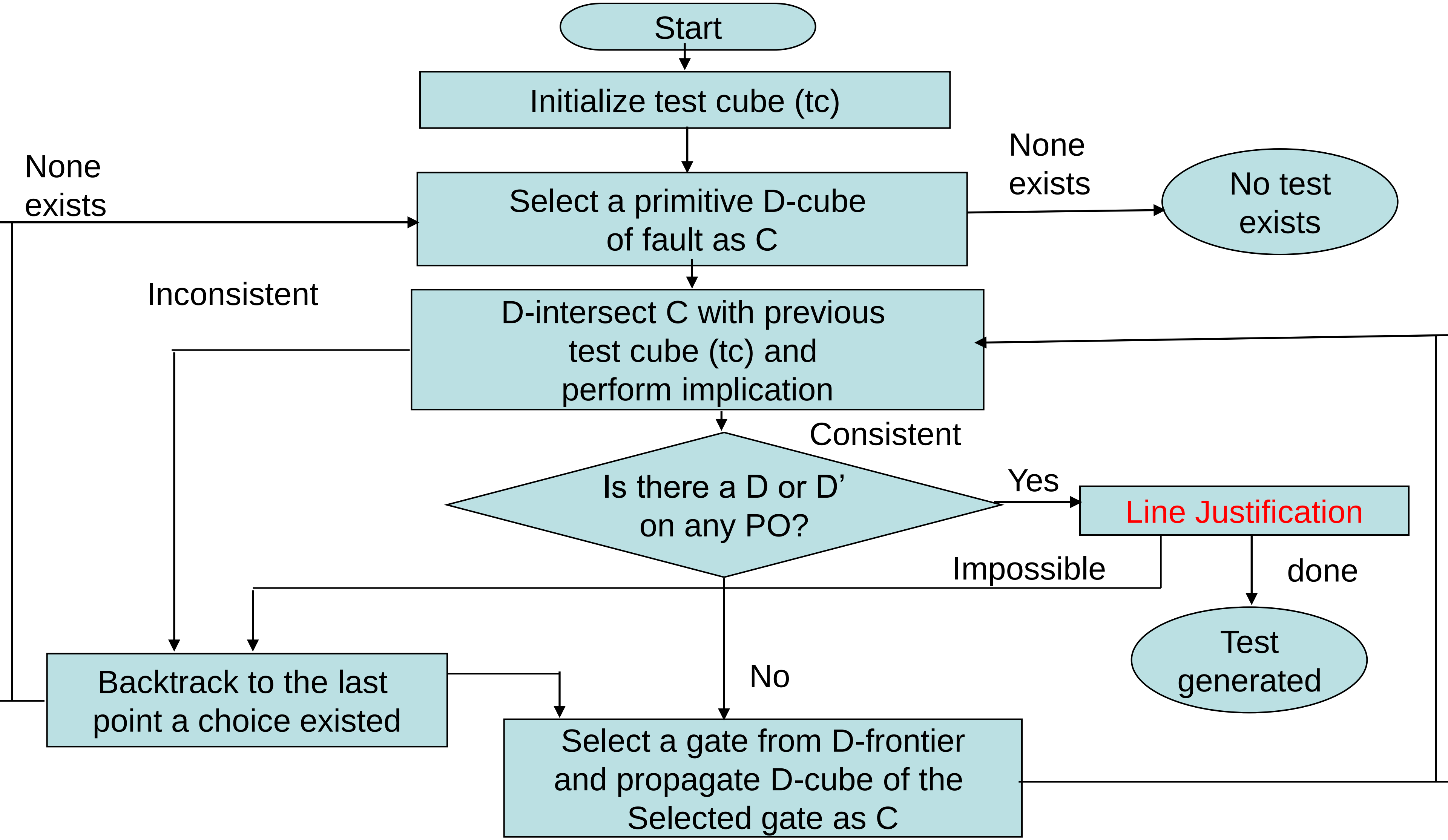
else pop prior implication stack setting to try alternate assignment;

if (no untried choices in implication stack)

fault untestable & stop;

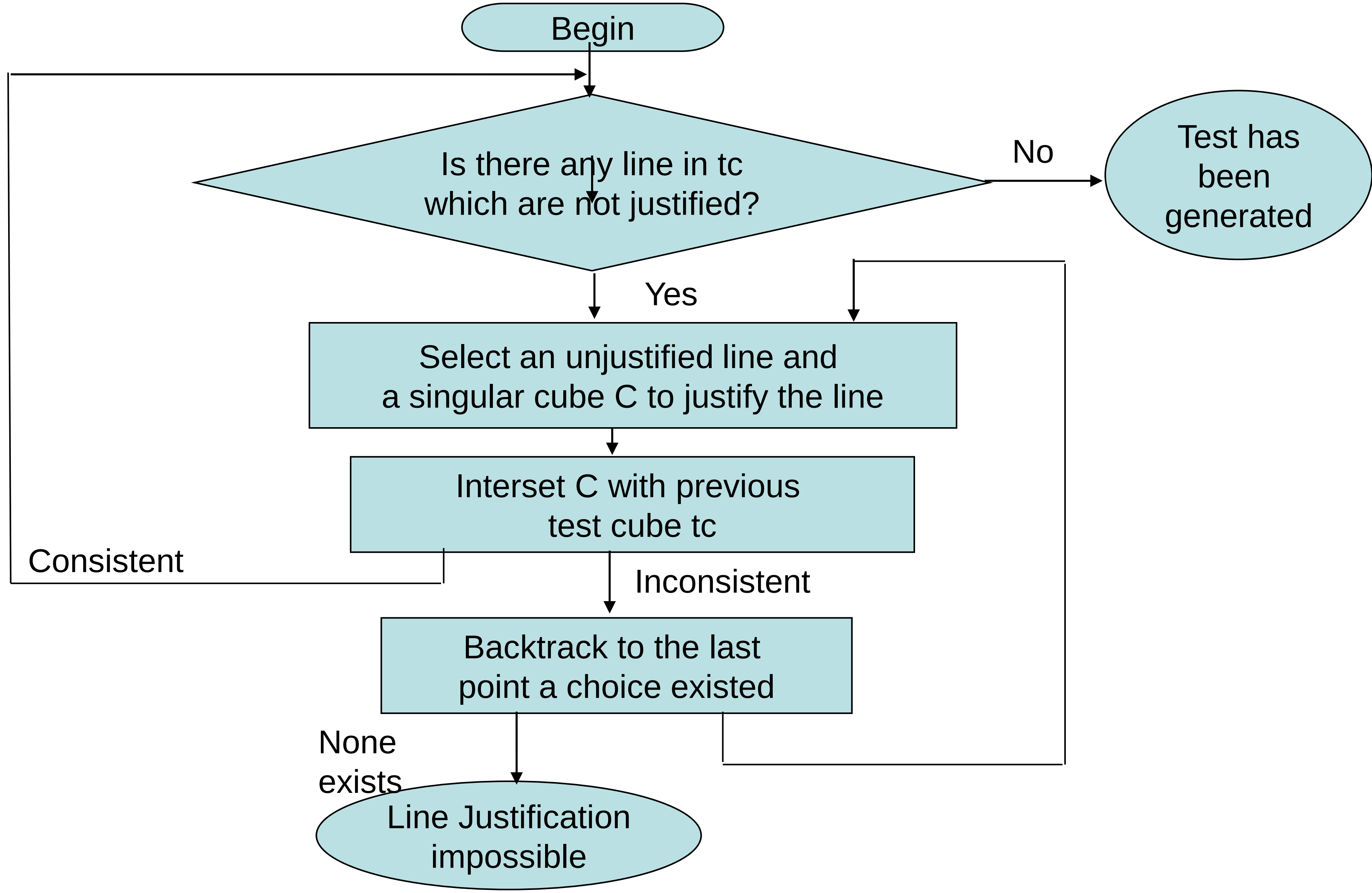
else return;

D-Algorithm

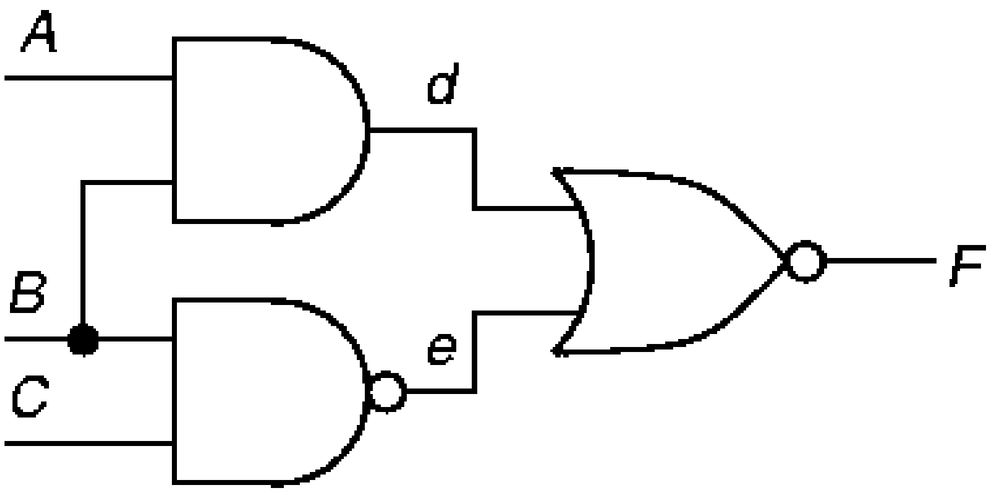


Start
Initialize test cube (tc)
None exists
Select a primitive D-cube of fault as C
None exists
No test exists
Inconsistent
D-intersect C with previous test cube (tc) and perform implication
Consistent
Is there a D or D' on any PO?
Yes
Line Justification
Impossible
done
Test generated
No
Backtrack to the last point a choice existed
Select a gate from D-frontier and propagate D-cube of the Selected gate as C

D-Algo (Line Justification)



Circuit Example1



Inputs			Output
A	*B*	*C*	*F*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Singular Cover & D-Cubes

A	B	C	d	e	F
1	1		1		
0			0		
	0		0		
	1	1		0	
	0			1	
		0		1	
				1	0
			1		0
			0	0	1
D	1		D		
1	D		D		
D	D		D		
	D	1		$\overline{D}$	
	1	D		$\overline{D}$	
	D	D		$\overline{D}$	
			D	0	$\overline{D}$
			0	D	$\overline{D}$
			D	D	$\overline{D}$

- ***Singular cover*** **– Used for justifying lines**
- ***Propagation D-cubes*** **– Conditions under which difference between good/failing machines propagates**

Steps for Fault *d* sa0

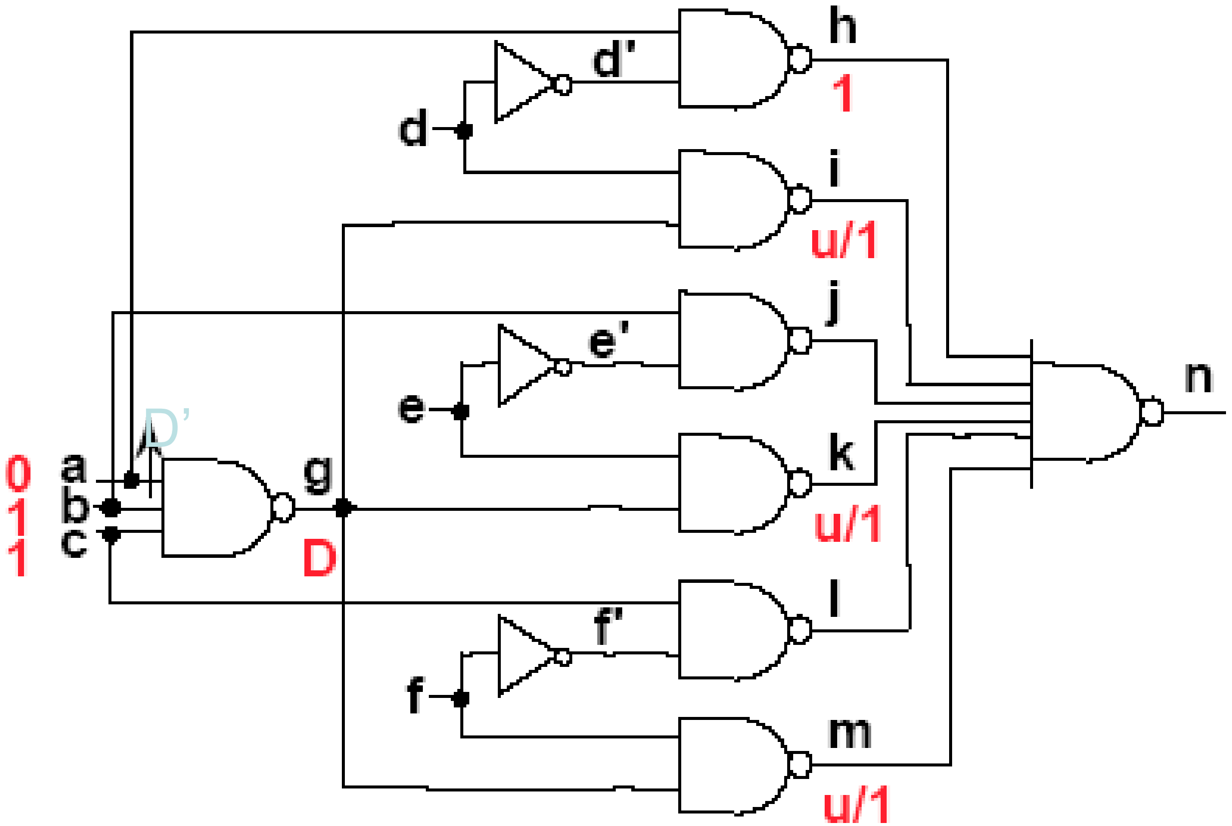
Step	*A*	*B*	*C*	*d*	*e*	*F*	Cube type
1	1	1		D			PDF of AND gate
2				D	0	$\overline{D}$	Prop. D-cube for NOR
3		1	1		0		Sing. Cover of NAND

9 - V Algorithm (Muth)

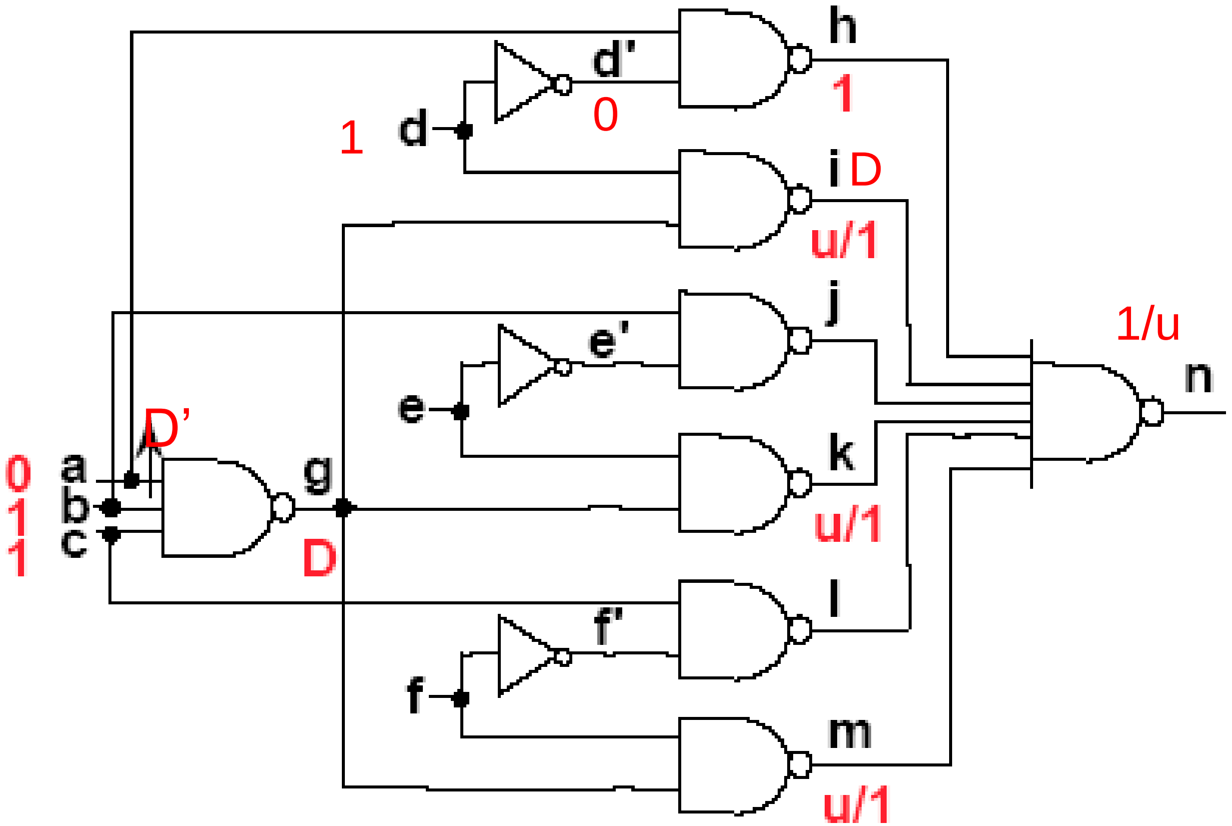
- **Logic values {0/0, 1/1, 0/1, 1/0, 0/u, 1/u, u/0, u/1, u/u}**
  - **0/u = {0, D'}, 1/u = {D, 1}, u/0 = {0, D}, u/1 = {D', 1}**
  - **u/u = {0, 1, D, D'}**

- **Reduces amount of search done in multiple path sensitization – D-Algo**

9 - V Algorithm



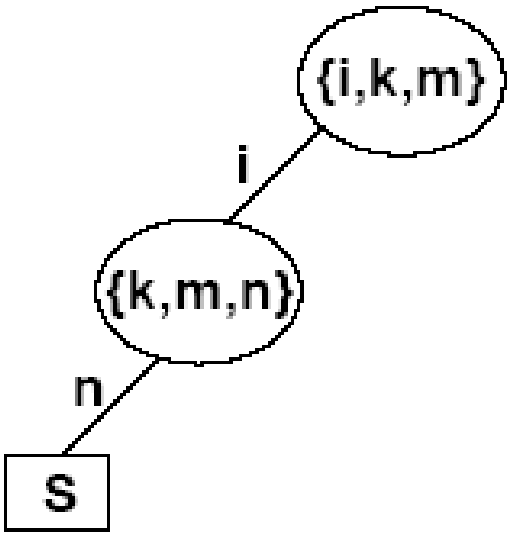
9 - V Algorithm



9-V Algorithm: Value Comp

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: Value Comp



No Backtracking

Path Oriented DEcision Making (PODEM)

P. Goel, IBM, 1981

Motivation

- IBM introduced semiconductor DRAM memory into its mainframes – late 1970's
- Memory had error correction and translation circuits – improved reliability
  - D-ALG unable to test these circuits
    - Search too undirected
    - Large XOR-gate trees
    - Must set all external inputs to define output
  - Needed a better ATPG tool

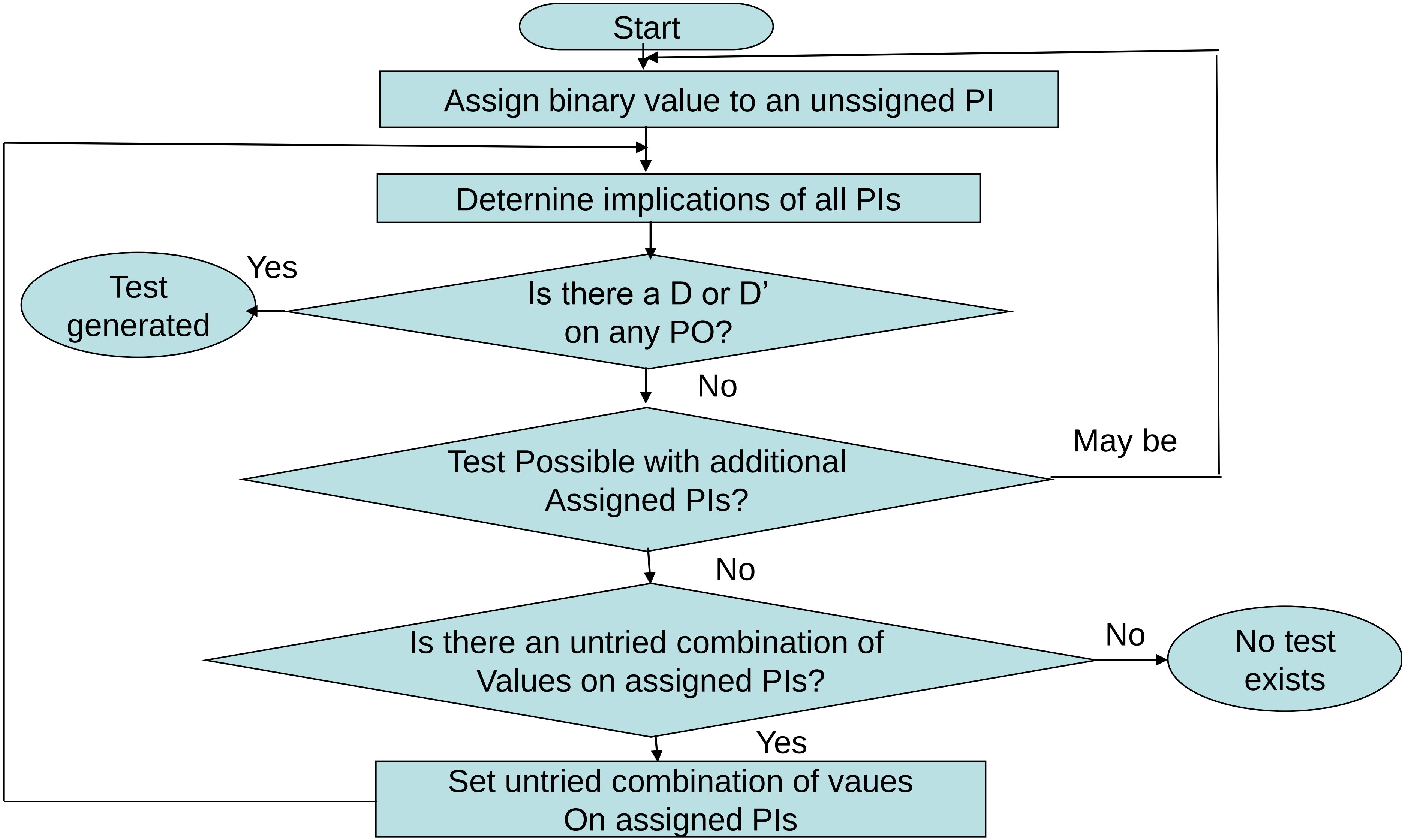
PODEM

- **New concepts introduced:**
  - **Expand binary decision tree only around primary inputs**
  - **Use *X-PATH-CHECK* to test whether *D-frontier* still there**
  - ***Objectives* -- bring ATPG closer to propagating D (D') to PO**
  - ***Backtracing***

PODEM High-Level Flow

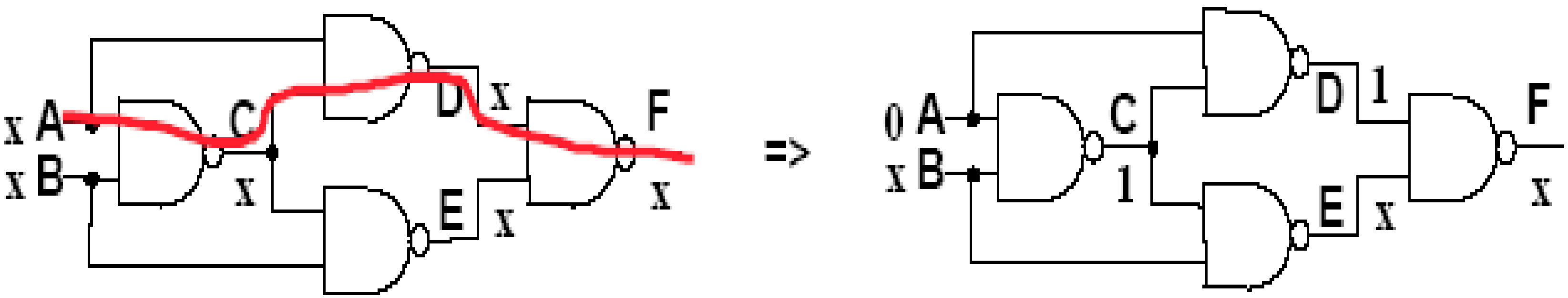
1. **Assign binary value to unassigned PI**
2. **Determine implications of all PIs**
3. **Test Generated? If so, done.**
4. **Test possible with more assigned PIs? If maybe, go to Step 1**
5. **Is there untried combination of values on assigned PIs? If not, exit: untestable fault**
6. **Set untried combination of values on assigned PIs using objectives and backtrace. Then, go to Step 2**

PODEM-Algorithm

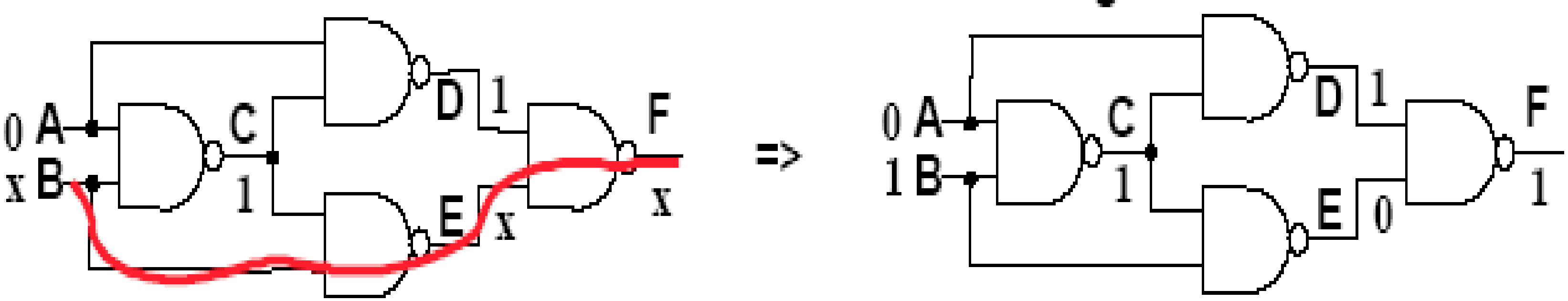


PODEM

Ex: Objective = (F,1).

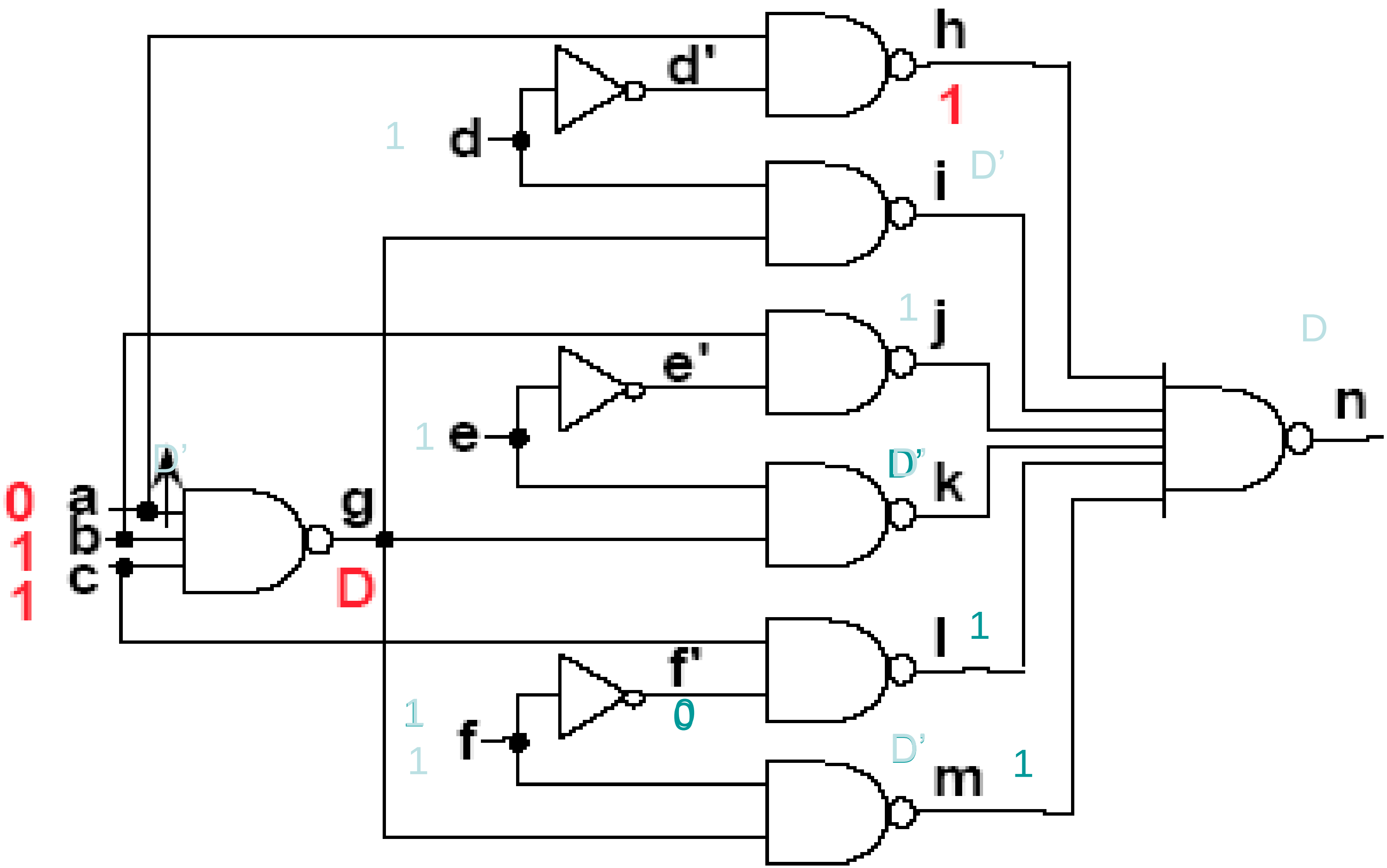


The first time of backtracing

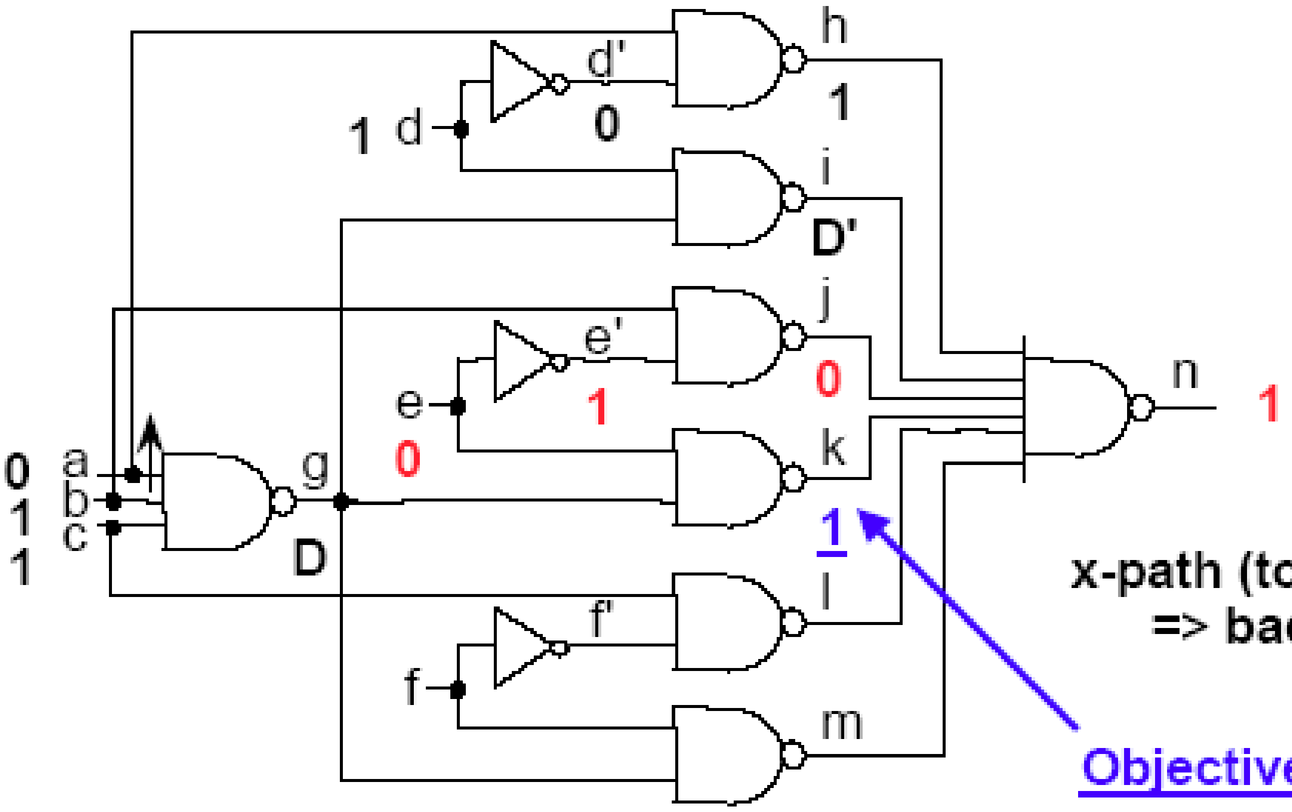


The second time of backtracing

D-Algorithm : Example



PODEM : Example



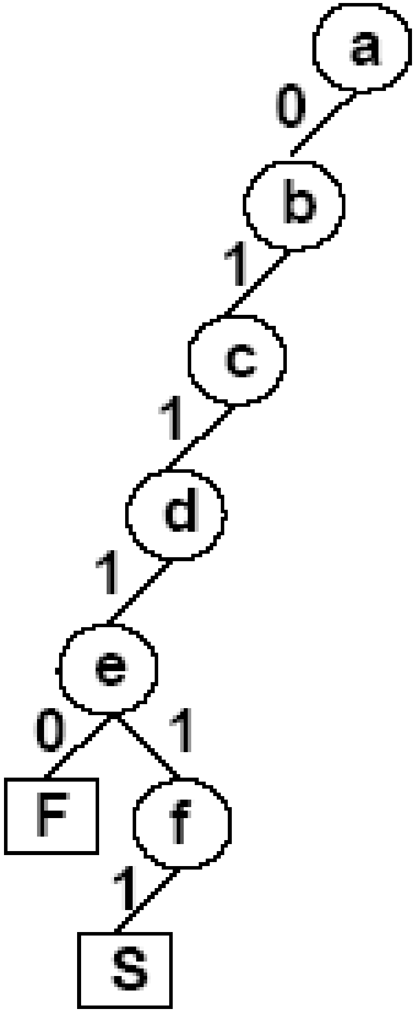
x-path (to PO) check fail
=> backtracking!!

Objective

PODEM : Value Comp

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



Thank You