Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

Module-IX

Lecture-I

Introduction to Automatic Test Pattern Generation (ATPG) and ATPG Algebras

Introduction

- In a circuit some faults are “easy to test” and some others are “difficult to test”.

- Algorithm to estimate the “easy to test faults” and the ones that are “difficult to test”.

- Fault simulation algorithms are used to determine test patters for easy to test faults and for the others, sensitization–propagation - justification approach is used.

- Automatic test pattern generation (ATPG) using sensitization–propagation -justification approach.
    - Basics of ATPG and ATPG algebras.
    - D-algorithm--the basic and first ATPG algorithm developed.

ATPG: Introduction

•ATPG stands for automatic test pattern generation.

•Procedure involves generation of input patterns that can ascertain presence or absence of fault(s) at some location(s) in a circuit.

(i) fault simulation and (ii) sensitization–propagation –justification.

These techniques are based on Boolean logic manipulations and are most widely used.

- ATPG technique that requires thermal imaging technique.
  - Images of the silicon (of the circuit) and then drawing conclusions based on temperature profile of various regions of the silicon.
  - For example, when a net say A, has stuck-at-0 fault and primary inputs are applied such that net A gets 1, then temperature in A will be abnormally higher, due to high short circuit current.

ATPG: Introduction

The advantage is, we need not have to worry about propagating the effect of the fault to primary outputs, as all internal points are observable in terms of temperature profile.

So, sensitization is enough to generate a test pattern.

However, the instrumentation cost for thermal imaging is high.
So, widely used ATPG algorithms are based on logic manipulations.

ATPG algebra

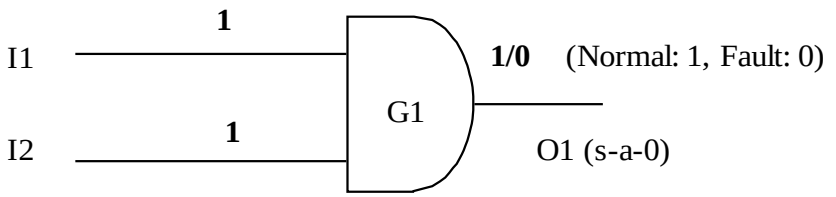
•Boolean algebra comprising of 0 and 1, is used for studying behavior of digital circuits.

•Circuits with fault: Boolean algebra will not suffice to represent signal values

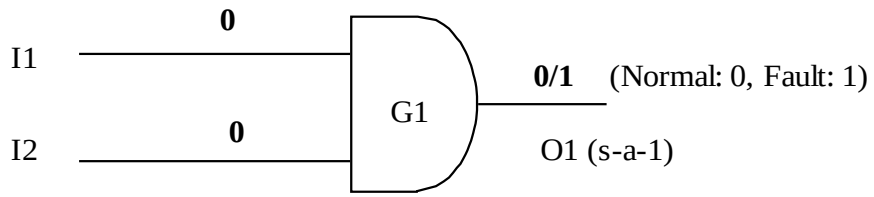
•I1=1,I2=1 detects s-a-0 fault at the output, we need to mark O1 as 1/0-indcating that under normal condition O1 is 1 and under fault it is 0.

•mark O1 as 0/1--indicating that under normal condition O1 is 0 and under fault it is 1.

•a single bit (0 or 1) is not enough for representing and studying circuits with fault.



(a) 1/0 is required to represent the detection of fault

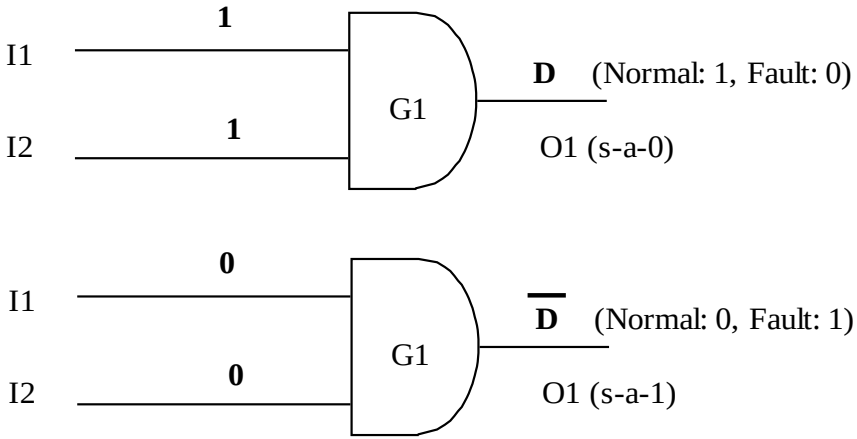


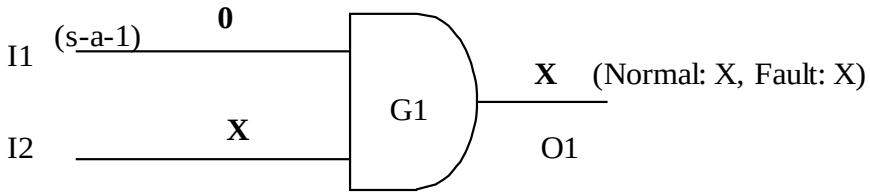
(b) 0/1 is required to represent the detection of fault

Roth's five value algebra

Symbol	Implication	Normal Circuit	Faulty Circuit
0	(0/0)	0	0
1	(1/1)	1	1
X	(X/X)	X	X
D	(1/0)	1	0
$\bar{D}$	(0/1)	0	1

Example to illustrate Roth's five value algebra





Example illustrate operations in Roth's five value algebra

- 1 AND D = D; 1 AND 1/0 involves two computations—(i) 1 AND 1=1, (ii) 1 AND 0=0.
- 0 OR $\overline{D}$ = $\overline{D}$; 0 OR 0/1 involves two computations—(i) 0 OR 0=0, (ii) 0 OR 1=1.
- NOT($\overline{D}$)=D; NOT(0/1) involves two computations—(i) NOT(0)=1, (ii) NOT(1)=0.

When one input is X in logical operation then output is also X, if others inputs are non controlling; else output is determined by the controlling input. Some examples are given below

- X AND 1 =X; 1 is non controlling in AND logic
- X AND 0 =0; 0 is controlling in AND logic

Types of ATPG algorithms

- **Exhaustive**

Testing by “exhaustive” technique implies applying all input combinations and checking for “functional” correctness. Exhaustive testing techniques are not used in ATPG because of complexity.

- **Random**

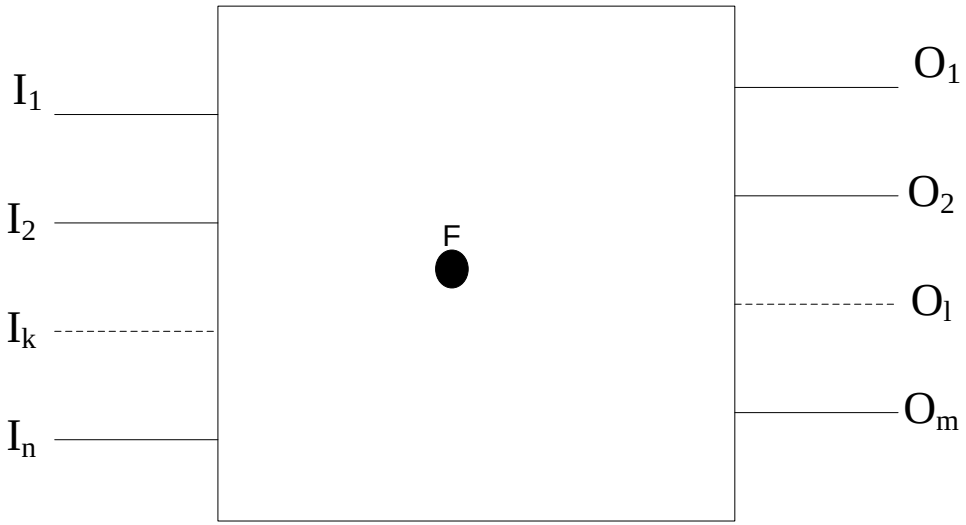
Random ATPG basically involves three steps:

- Generate a random pattern
- Determine how many faults are detected by the random pattern (fault simulation)
- Continue the above two steps till no (or few) new faults are detected by the random pattern.

- Deterministic ARPG techniques.
  - Symbolic difference
  - Path sensitization.

Symbolic difference Based ATPG

ATPG by symbolic difference is basically based on well-known Shannon's expansion theorem. Let us explain this by an example. Consider an arbitrary Boolean function as $f(x_1, x_2, ..x_n)$. By Shannon's expansion rule, the function can be expanded about any variable x_1 say, as $f(x_1, x_2, ..x_n) = \overline{x_1}.f(0, x_2, ..x_n) + x_1.f(1, x_2, ..x_n)$.



Block diagram of a circuit with a fault at point F

Symbolic difference Based ATPG

Let the following Boolean functions hold

- $g = f_F(I_1, I_2, ..I_n)$ be the function for the Boolean value at net F (in terms of all the inputs)
- $o_i = f_i(g, I_1, I_2, ..I_n),\ 1 \le i \le m$ be the functions for output lines O_1 to O_m, under failure at point F. It may be noted that without fault all the output lines are functions of the inputs. With the failure, the output lines are functions of the inputs and the Boolean value at point F.

  Now, Boolean difference, or Boolean partial derivative, of the circuit block output $f_i, 1 \le i \le m$ with respect to g is:

$$\frac{\partial f_i}{\partial g} = f_i(1, I_1, I_2, ..I_n) \oplus f_i(0, I_1, I_2, ..I_n)$$

  For detecting s-a-0 fault at location F, we need

1. $g = f_F(I_1, I_2, ..I_n) = 1$
2. $(\exists j, 1 \le j \le m) \wedge \left(\frac{\partial f_j}{\partial g} = f_j(1, I_1, I_2, ..I_n) \oplus f_j(0, I_1, I_2, ..I_n) = 1 \right)$

Symbolic difference Based ATPG

Similarly, for detecting s-a-1 fault at location F, we need

1. $g = f_F(I_1, I_2, ..I_n) = 0$

2. $(\exists j, 1 \le j \le m) \wedge \left(\frac{\partial f_j}{\partial g} = f_i(1, I_1, I_2, ..I_n) \oplus f_i(0, I_1, I_2, ..I_n) = 1 \right)$

Unfortunately, due to high complexity the Boolean difference is not an efficient way to compute test patterns for large circuits.

Path Sensitization Based ATPG

ATPG by path sensitization method is generally applied for “difficult to test faults” and comprises three phases.

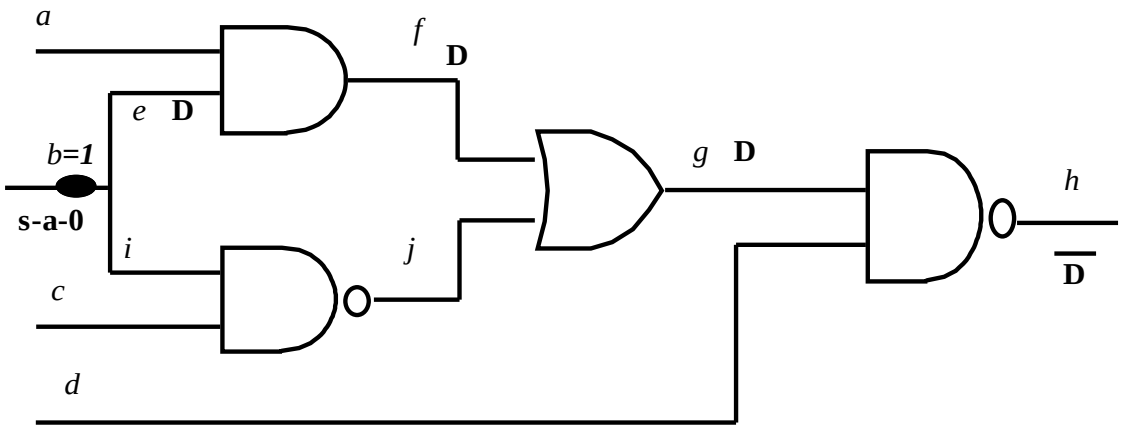
Fault sensitization: In this step a stuck-at fault is activated by setting the signal driving the faulty net to an opposite value from the fault value.

Fault propagation: In this step a path is selected from the fault site to some primary output, where the effect of the fault can be observed for its detection.

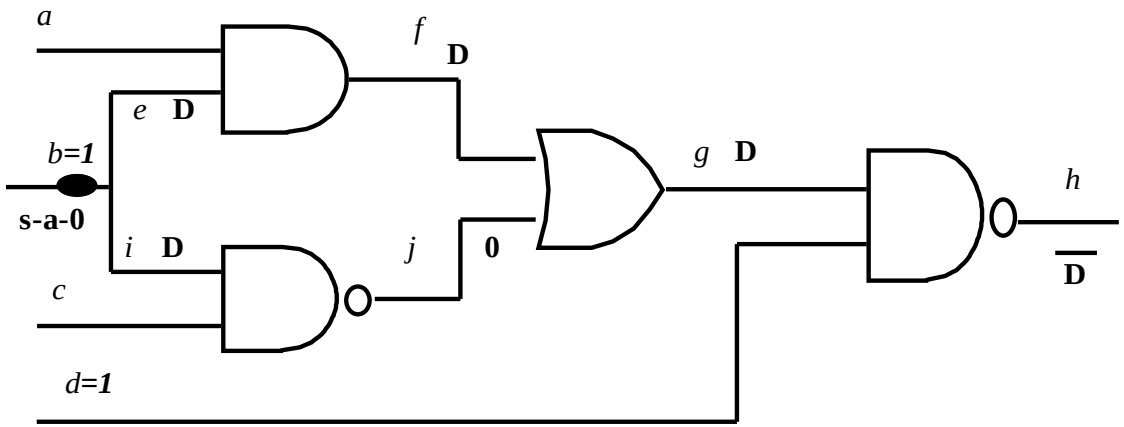
Line justification: In this step the signals in (internal) nets or some primary inputs, which were assigned for fault sensitization/propagation, are justified by setting (remaining) primary inputs of the circuit.

In the second and third steps, a conflict may occur, where a necessary signal assignment contradicts some previously-made assignment. When conflicts occur we need to take a new alternative path for fault propagation and see if all signals can be justified.

Path Sensitization Based ATPG: Example

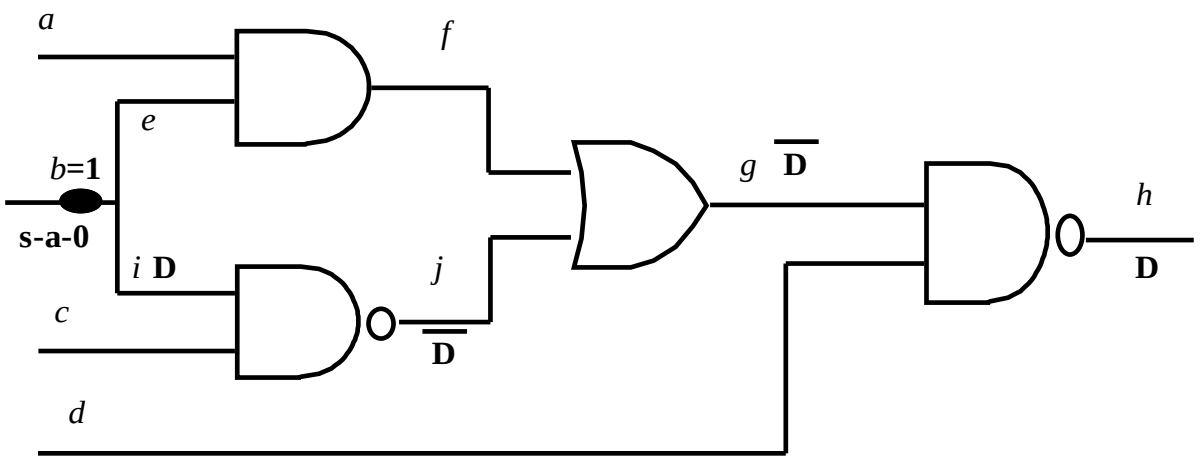


(a) Sensitization by *b*=1
Propagation by path e-f-g-h

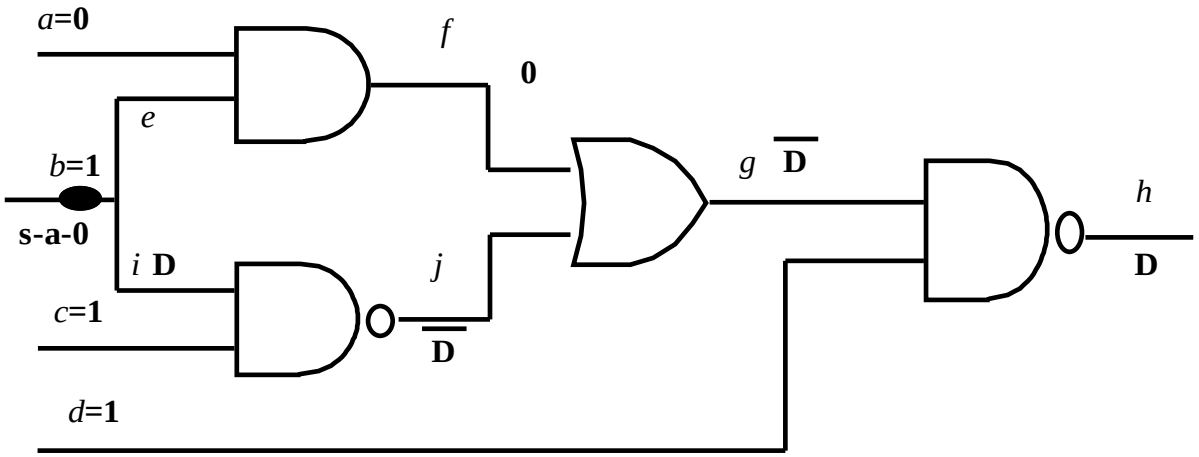


(b) Justification j=0
j*=D, *if c=1 and j=1 if c=0
(conflict at j)

Path Sensitization Based ATPG: Example



(a) Sensitization by *b*=1
Propagation by path i-j-g-h

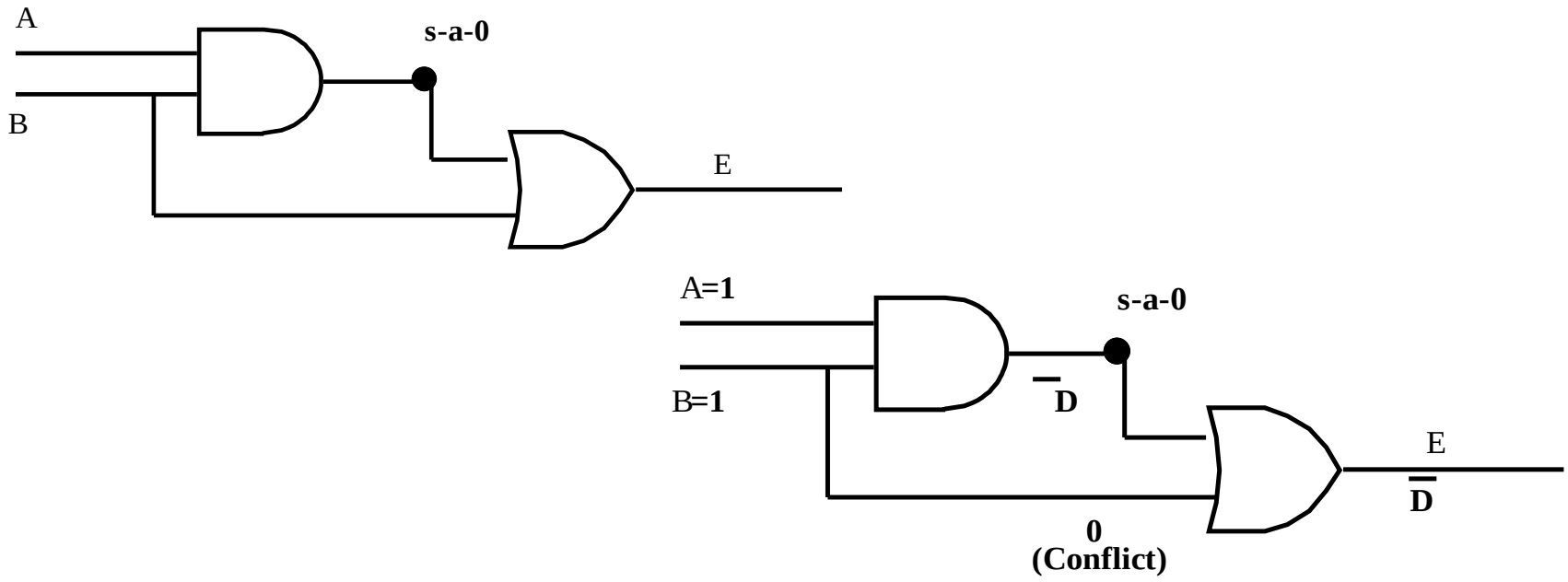


(b) Justification *d*=1,*c*=1,*f*=0,*a*=0
(Successful)

Questions and Answers

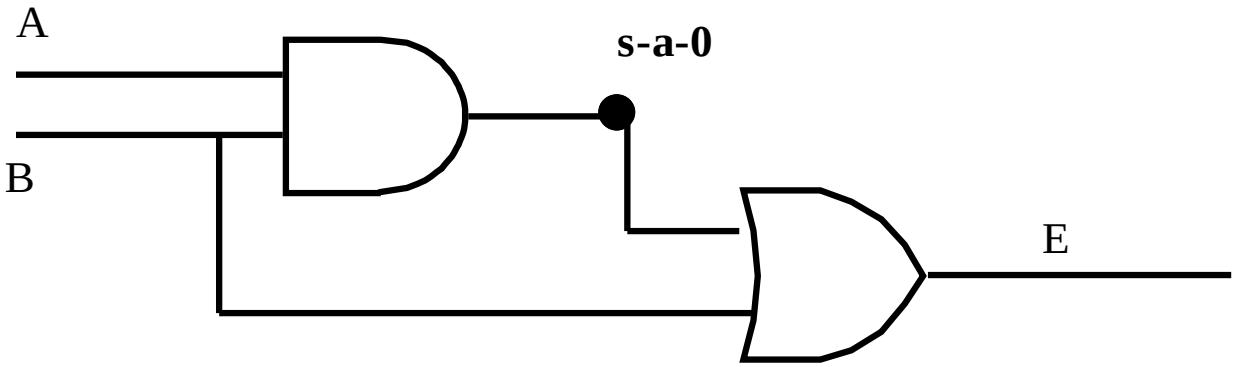
Question: Using path sensitization and Routh's algebra generate test pattern for the s-a-0 fault in the circuit given below:

Answer: To sensitize the fault, 1 is to be applied at the output of the AND gate. Also, there is only one path for fault propagation-- fault location to E. Now, to justify we need to apply a 0 at the second input of the OR gate. This leads to a collision. For justification we need a 0 in the second input of the OR gate and to sensitize the fault B is to be 1, which makes the second input of the OR gate a 1. This is illustrated in the figure below in terms of Routh's algebra.



Questions and Answers

Answer: Now, since there is no other alternative path for fault propagation, this fault is not testable. In fact the answer is true. This fault is not testable. This example actually illustrates another utility of ATPG algorithm—finding redundancy in circuits. If we observer carefully, function represented by this circuit is: E=A.B+B. This can be simplified as B(A+1), which is B. So these two gates are redundant. So, if ATPG by path sensitization discovers that a fault is not testable, then there is redundancy in the circuit.



Thank You

Design Verification and Test of Digital VLSI Circuits NPTEL Video Course

Module-IX

Lecture-II and III

D-Algorithm

Introduction

- Roth's D-Algorithm [1] in depth, which is the first formal algorithm for ATPG.
  - Uses Routh's algebra, which is suitable for ATPG using the "sensitize-propagate-justify" approach.

- Basic definitions and procedures required in D-algorithm.

- Issues in D-algorithm, and how they were addressed in some advanced algorithms.

D-Algorithm: Definitions and procedures

Definition 1: Singular cover

Singular cover of a logic gate is the minimal set of input signal assignments needed to represent essential prime implicants in the Karnaugh map of the logic gate, both for the case 0 and case 1.

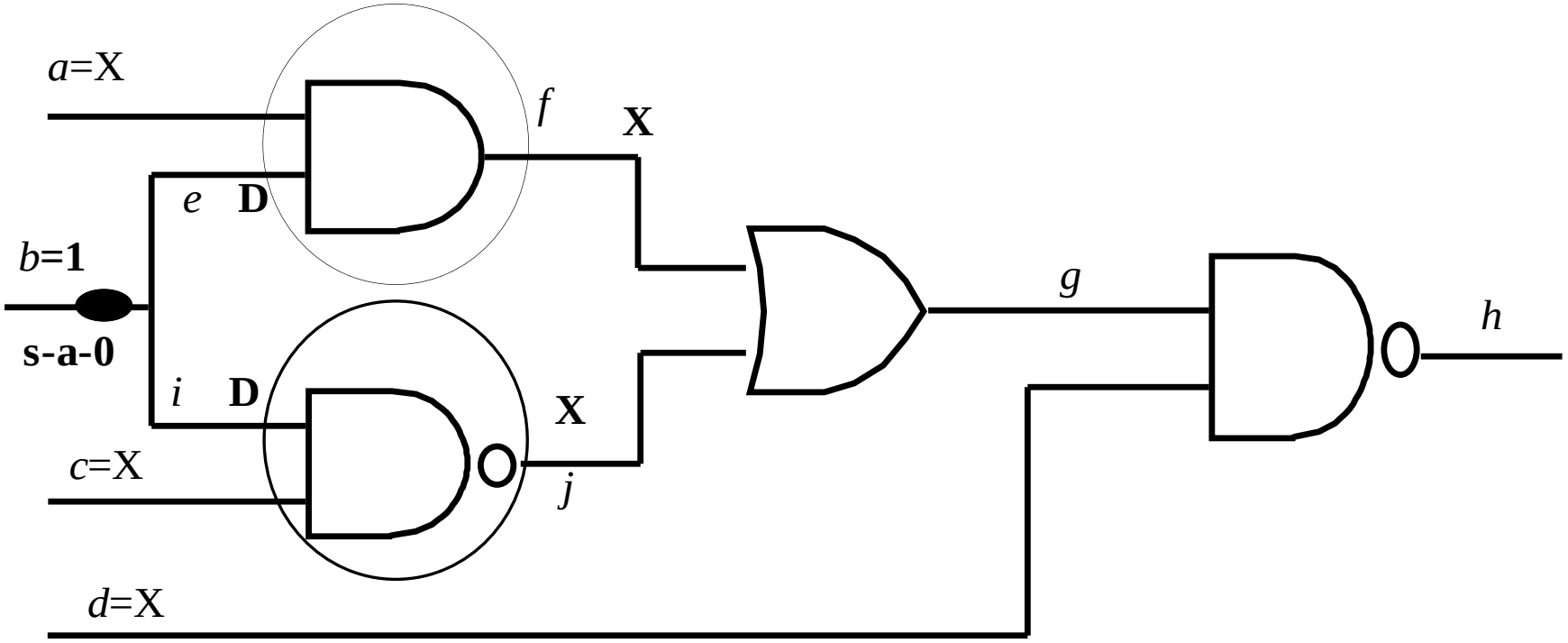
Table shows singular cover for 2 input AND and 2 input OR gate.

AND	Inputs A	B	Output		OR	Inputs A	B	Output
	0	X	0			1	X	1
	X	0	0			X	1	1
	1	1	1			0	0	0

D-Algorithm: Definitions and procedures

Definition 2: D-frontier

The D-frontier comprises gates whose output value is X and at least one of its input is D or $\bar{D}$. This implies that, any gate in D-frontier can be used for fault propagation.



D-Algorithm: Definitions and procedures

Definition 2: D-frontier

The basic idea is, X at the inputs can be appropriately selected so that D or $\bar{D}$ can be propagated to the output of the gates.

In other words, faults can be propagated only through gates in the D-frontier.

Once the fault is propagated the gate is deleted from the D-frontier list.

D-Algorithm: Definitions and procedures

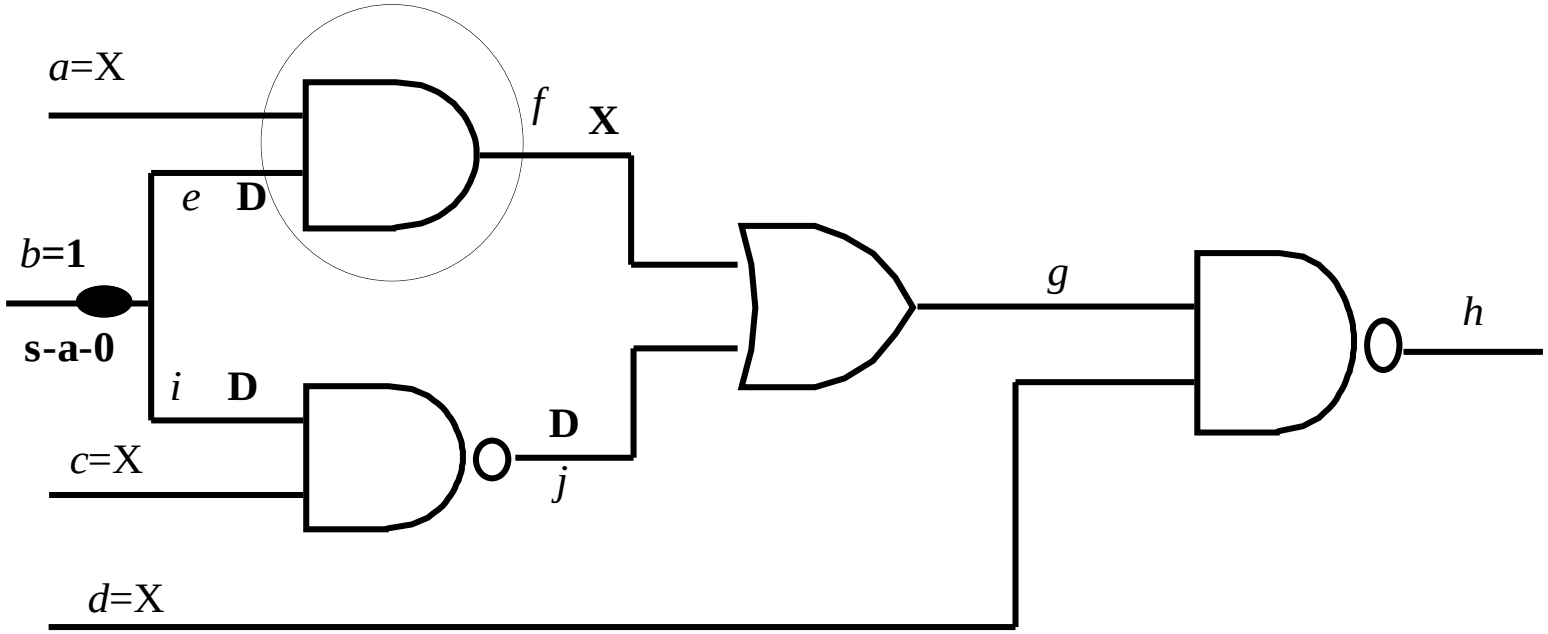
Definition 3: Unique D-Drive

If there is only one gate in D-frontier, then fault effect has to be propagated through that gate. This situation is called unique D-drive.

D-Algorithm: Definitions and procedures

Definition 4: J-frontier

The J-frontier comprises gates whose output value is known (0 or 1) but its inputs are not yet computed (or not yet implied by its inputs). In other words, output of a gate in the J-frontier is known, but inputs are not yet computed. This implies that, any gate in J-frontier can be used for justification.



D-Algorithm: Definitions and procedures

- Assume that it was decided that fault effect D be propagated via *j.*
  - So, *j*=D by forward implication.
  - Also, *f*=0 for propagating D to *g*.

- Now, the encircled gate is in the J-frontier, because the output is known as 0 and its inputs are X, which can be decided in *justification* step.

- The basic idea is, X at the inputs can be appropriately selected so that output is justified. In other words, during justification gates in J-frontier can only be considered. Once the inputs are justified, the gate is deleted from the J-frontier list.

D-Algorithm: Definitions and procedures

Procedure 1: Implication

The implication procedure comprises 3 steps

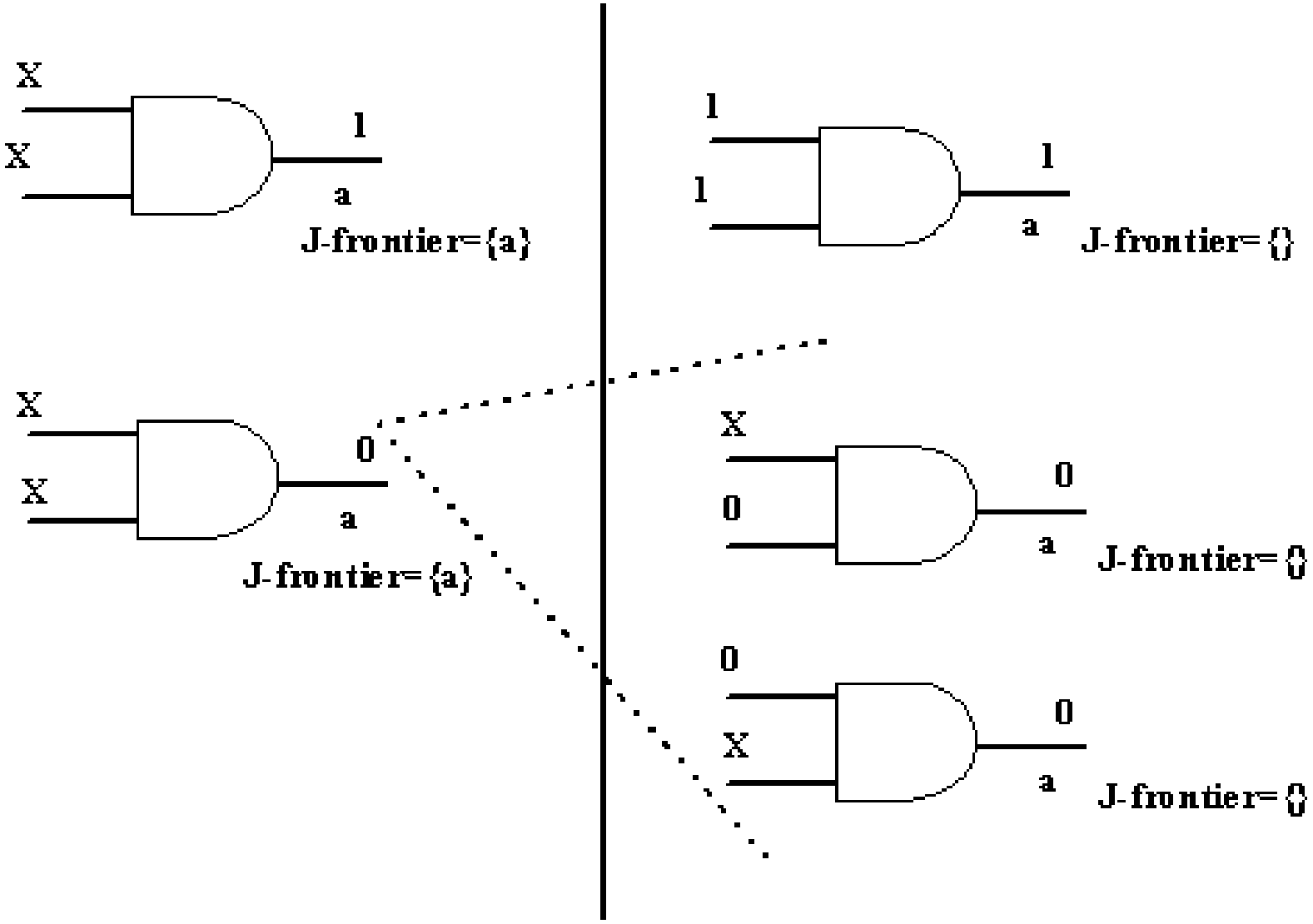
1. Compute all the values of the signals that can be determined uniquely from already given signal values (of some nets). If many choices are available, for example as in singular cover, any one of them can be considered.
2. Maintain J-frontier and D-frontiers
3. Check for consistency and stop in case of inconsistency (i.e., contradictory signal values are implied by the implication procedure).

Basically, implication procedure is similar to a simulation process where values of all nets (and finally primary outputs) are determined starting from primary inputs.

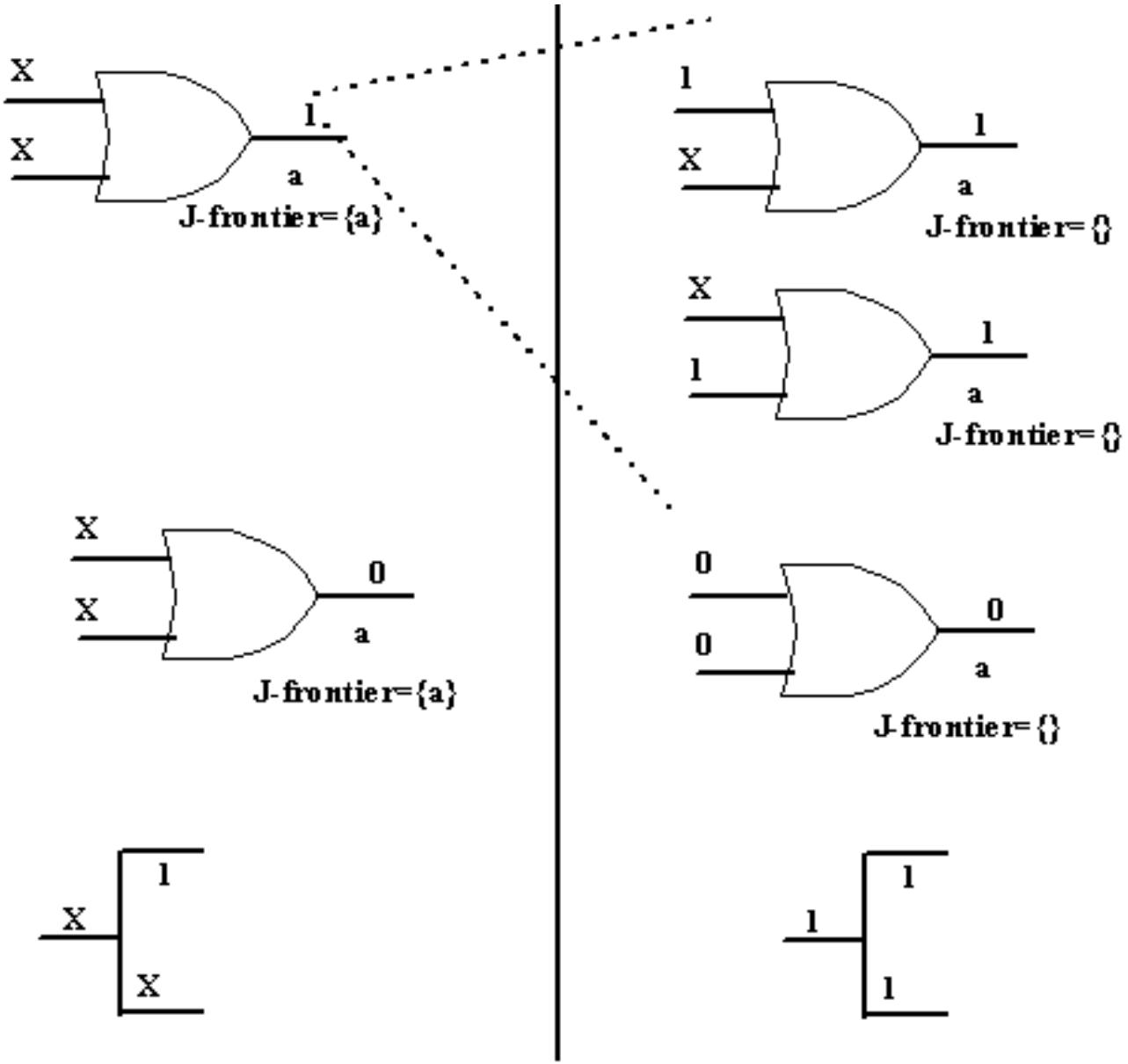
D-Algorithm: Simulation and Implication

Simulation	Implication
All the signal values are determined uniquely	All signals may not be determined uniquely
Value assignment moves from inputs to outputs of a circuit	Signal assignments propagate both towards primary inputs and primary outputs. For example, to test a s-a-1 fault at net *l* say, we need to have implication in two directions (i) backwards, to primary inputs to make *l*=0 and (ii) forward, to primary outputs to propagate D.
There is no inconsistency	Inconsistency may arise, when for a given net *l* (which is not the fault location) different signal values (0 or 1) need to be assigned.

D-Algorithm: Simulation and Implication



D-Algorithm: Simulation and Implication

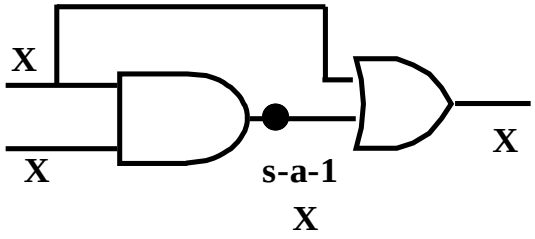


D-Algorithm: Simulation and Implication

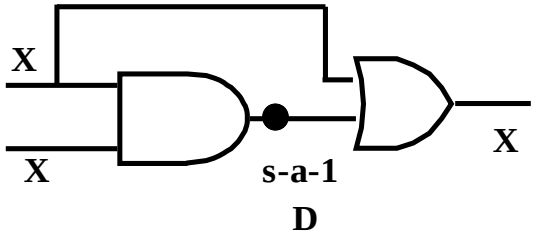
The second gate has output of 0 and so by singular cover we have two options for assigning values to the inputs; the two options are shown in the right side.

It may be noted that we could have also kept another option where both the inputs are kept 1. However, this is avoided because if both inputs are fixed then flexibility gets reduced and chances of inconsistency increases.

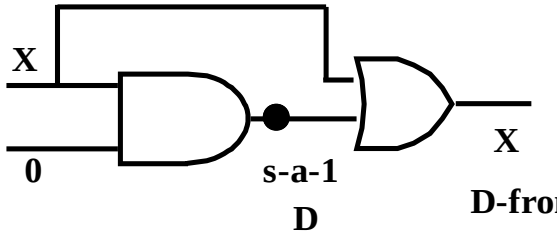
D-Algorithm: Simulation and Implication



Step-0

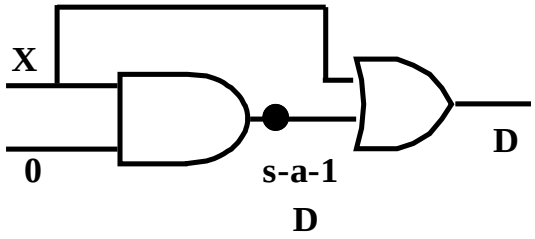


Step-1 (unique D-drive)

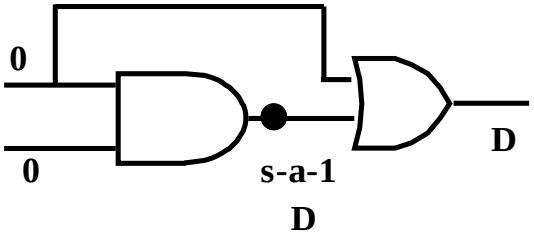


D-frontier

Step-2 (Backward implication)

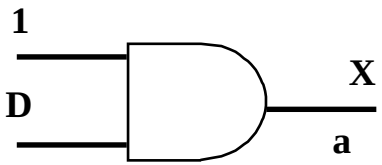


Step-3 (Propagate D through D-Frontier)

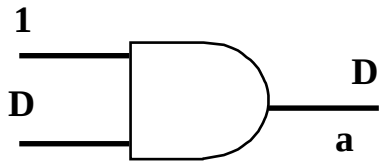


Step-4 (Backward Implication)

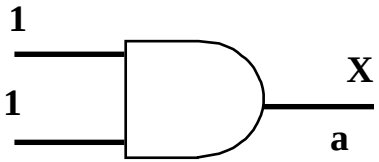
D-Algorithm: Forward Implication and D-frontier

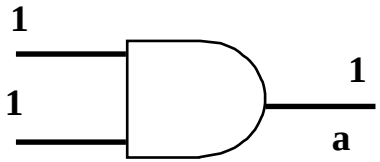


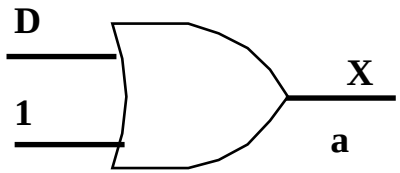
D-frontier={a}



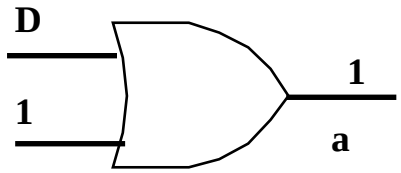
D-frontier={}



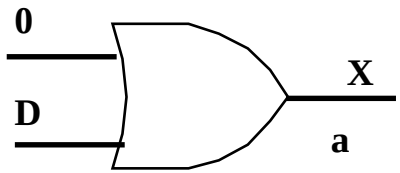




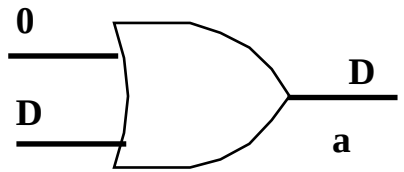
D-frontier={a}



J-frontier={}



D-frontier={a}

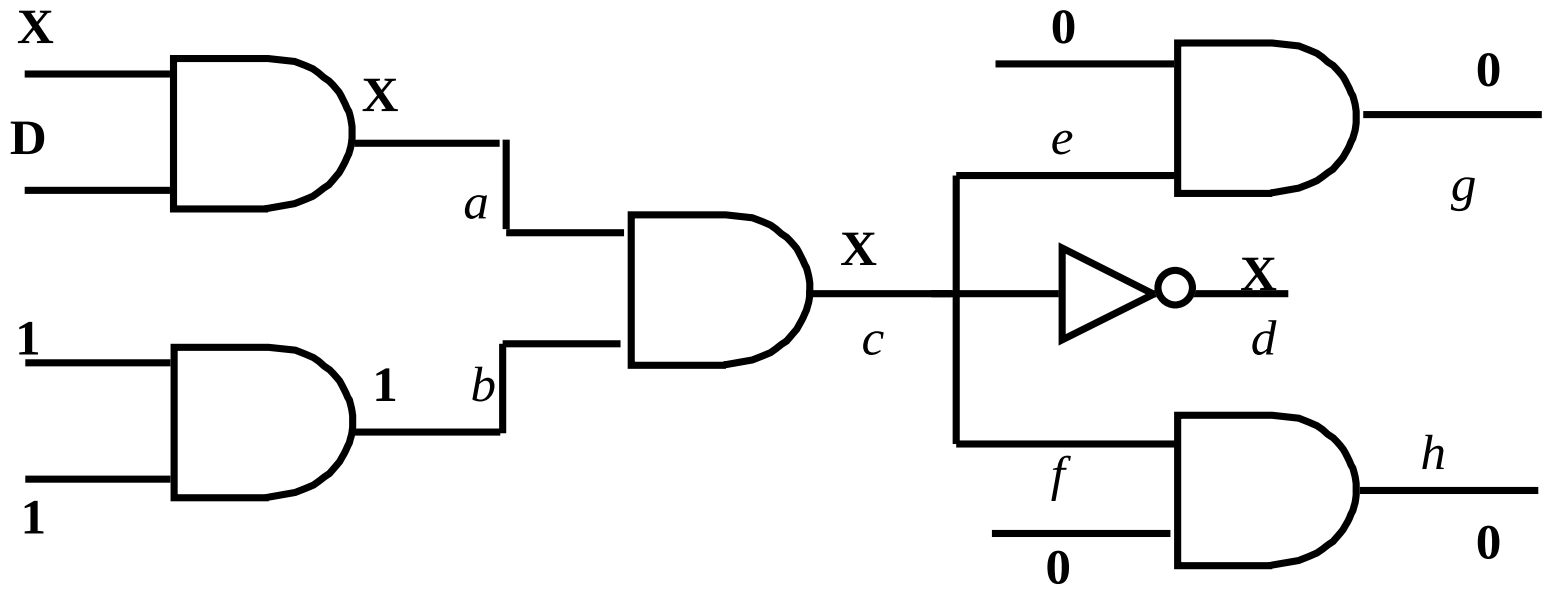


D-frontier={}

D-Algorithm: Definitions and procedures

Procedure 2: X-path

An X-path is a path of consecutive nets in a circuit all of whose values are X. Let A be a gate in a D-frontier. The faults on the inputs of A can be propagated to a primary output O only if there is an X-path from A to O. In Figure below there is only one X-path from *a* to output—*a-c-d.*



D-Algorithm

Input: Circuit netlist and one stuck at fault with its location.

Output: Primary input values and expected values at the primary outputs

Step-1: Initialize all the signal values of the circuit to X.

Step-2: Sensitize the fault location, i.e., if s-a-0 is the fault, apply D in the fault location, else if s-a-1 is the fault, apply $\overline{D}$ in the fault location.

Step 3: Determine/update gates in D-frontier and J-frontier, due to signal changes from X to 0,1, D or $\overline{D}$. If there is no gate in D-frontier and D /$\overline{D}$ has not reached any primary output, backtrack().

Design Verification and Test of Digital VLSI Circuits
NPTEL Video Course

Module-IX

Lecture-III

D-Algorithm

D-Algorithm

Step-4: If D /$\overline{D}$ has not reached any primary output, propagate D /$\overline{D}$ through one of the gates in the D-frontier.

Step-5: Use forward and backward implications to determine as many signal values as possible. backtrack() in case of an inconsistency.

Step-6: If D /$\overline{D}$ has reached any primary output and the primary inputs have received values by backward implications, stop; the values of the primary inputs comprise the test pattern. Else, go to Step-3.

D-Algorithm

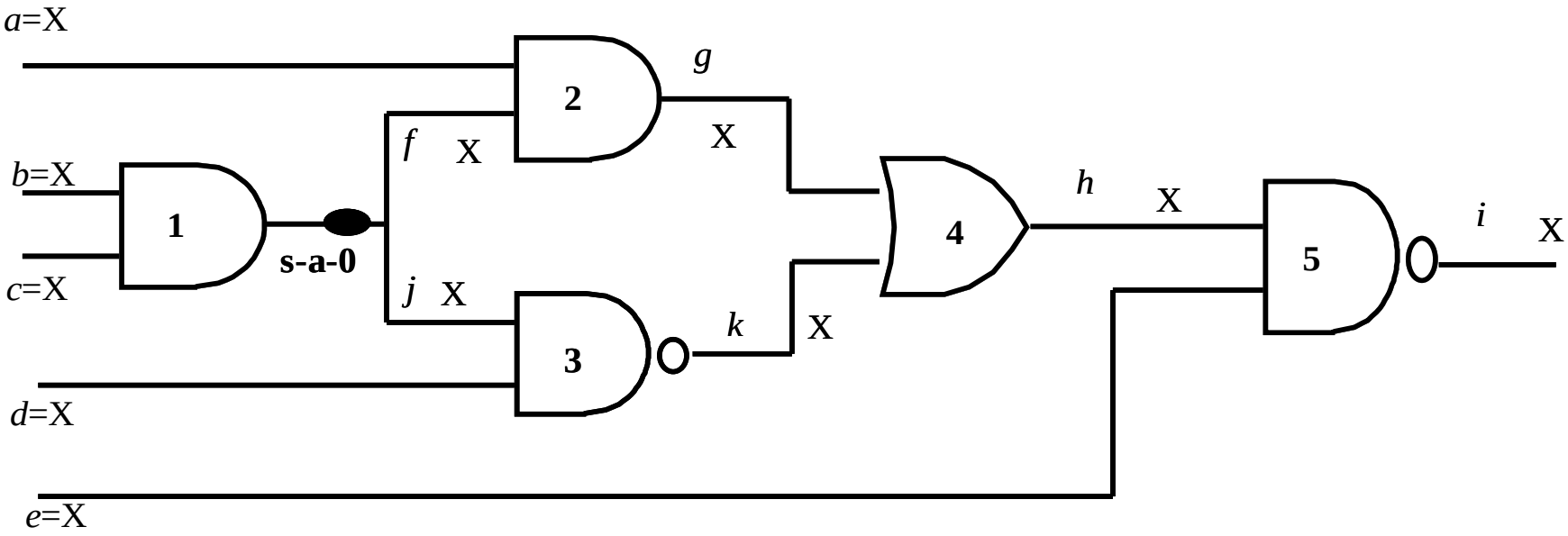
The module backtrack(), works as follows.

Step-1: Determine the last choice (of alternatives) taken in selection of gates in D-frontier or signal values in forward/backward implications, where another new alternative is available. Take any one of the other alternative. If no choice is left, ***report non testable fault*** and ***return***.

Step-2: Undo all steps (i.e., propagation of D / $\overline{D}$ and forward/backward implications), from the current signal values to point where another alternative is attempted. **Return**.

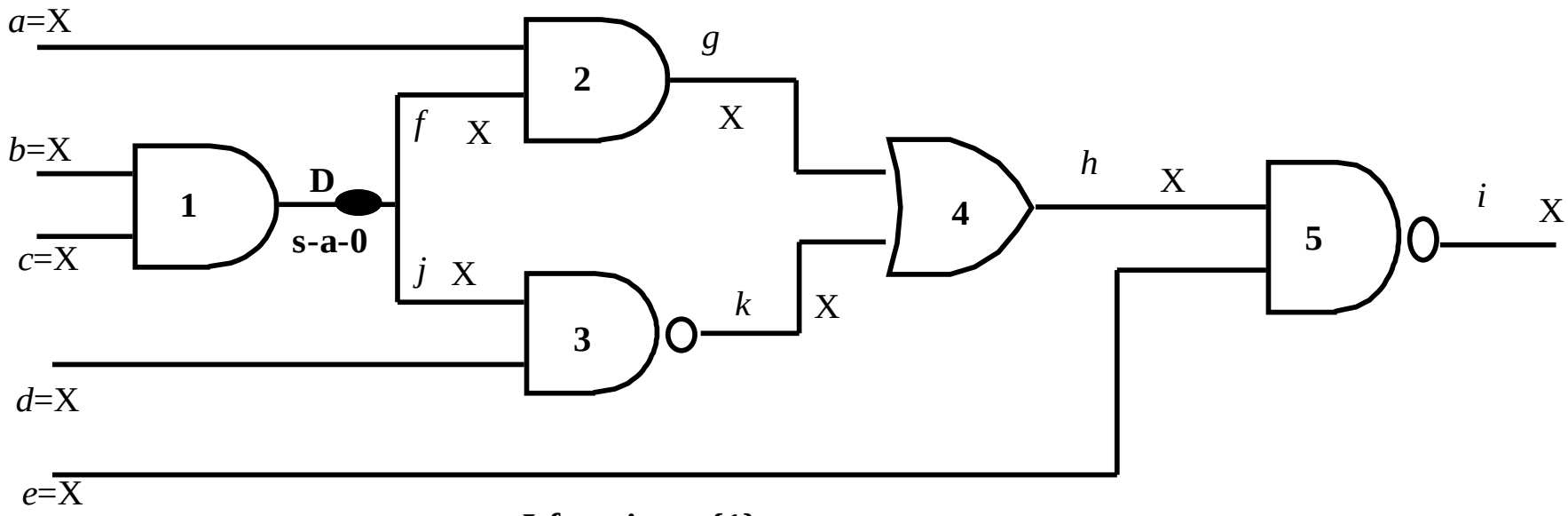
D-Algorithm: Example

Example of simple digital circuit with a s-a-0 fault



D-Algorithm: Example

Sensitizing the fault location and computation of J- and D-frontier

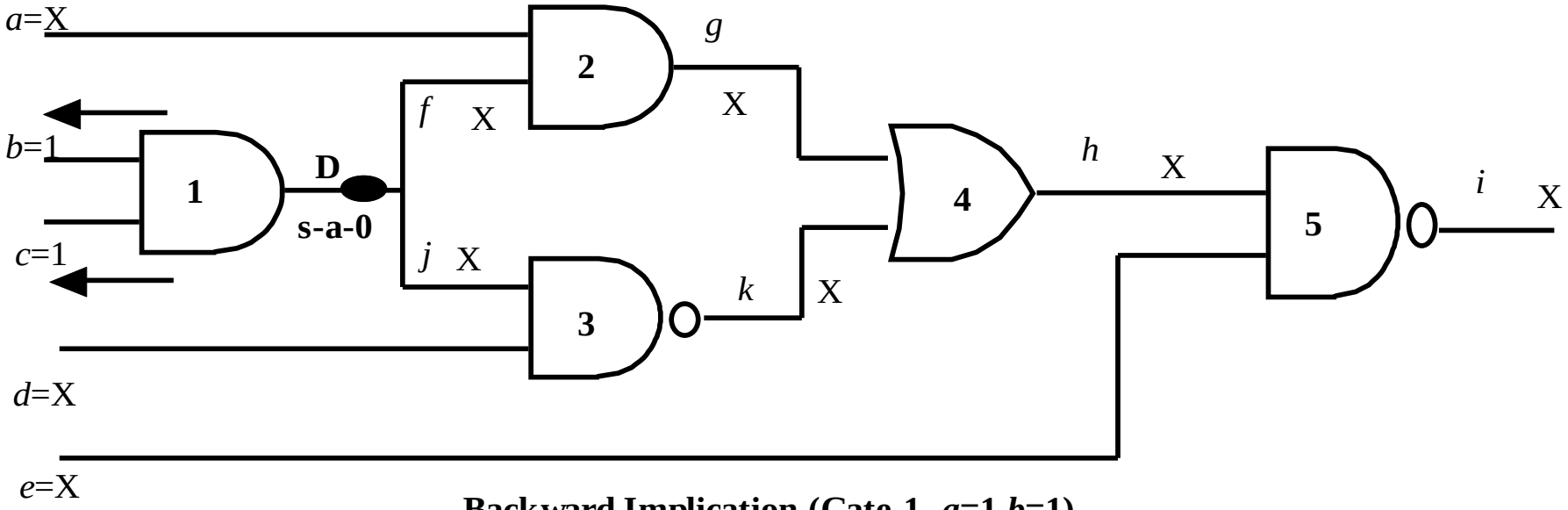


J-frontier = {1}

D-frontier = {2,3}

D-Algorithm: Example

Backward implication at gate-1

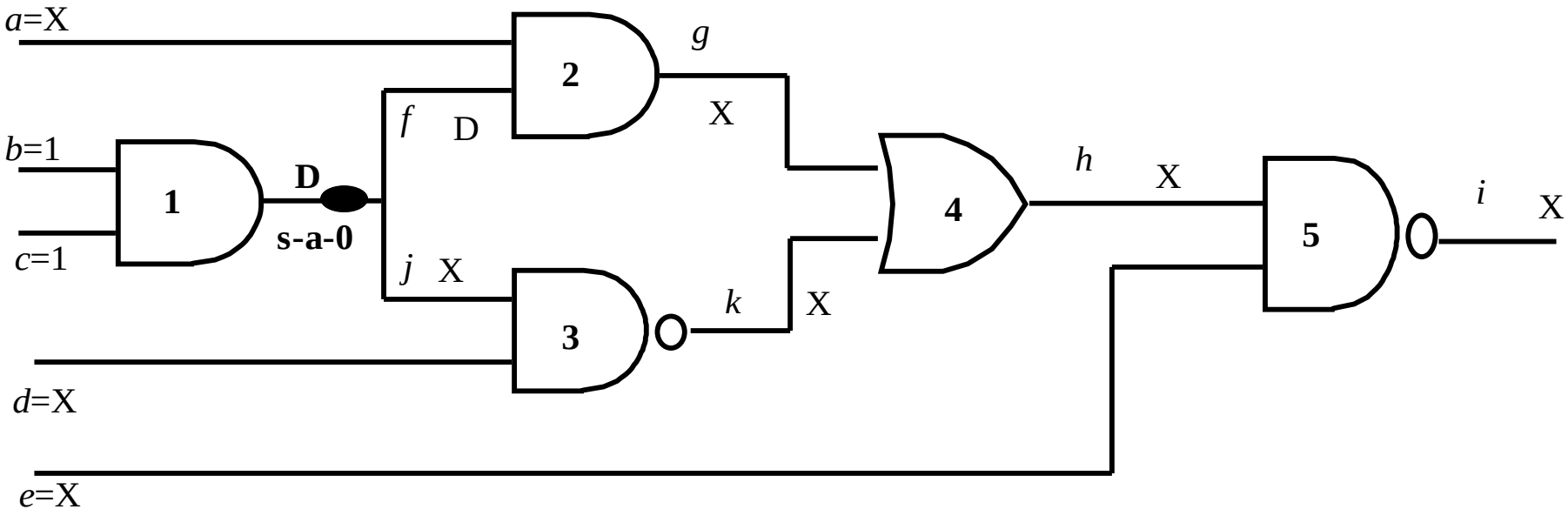


Backward Implication (Gate 1, *a*=1,*b*=1)

J-frontier = {}

D-Algorithm: Example

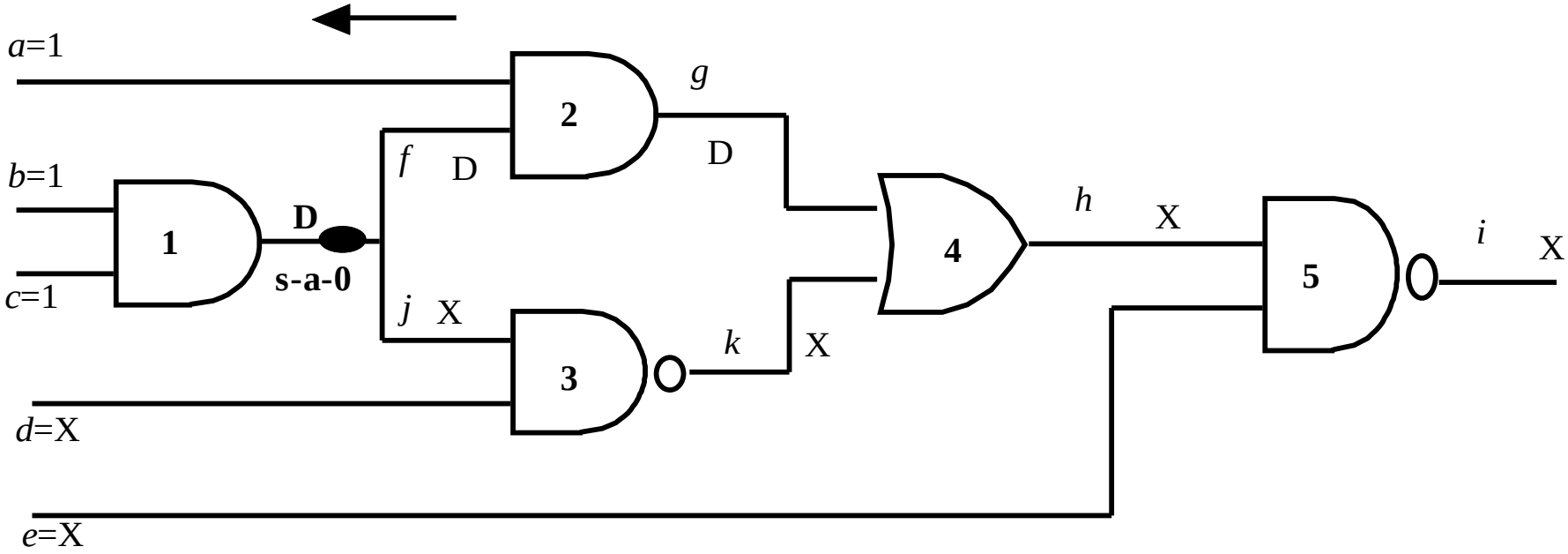
Selection of gate-2 as D-frontier



Possible D-frontier = {2,3}, select 2 for D propagation
Modified D-frontier ={2}

D-Algorithm: Example

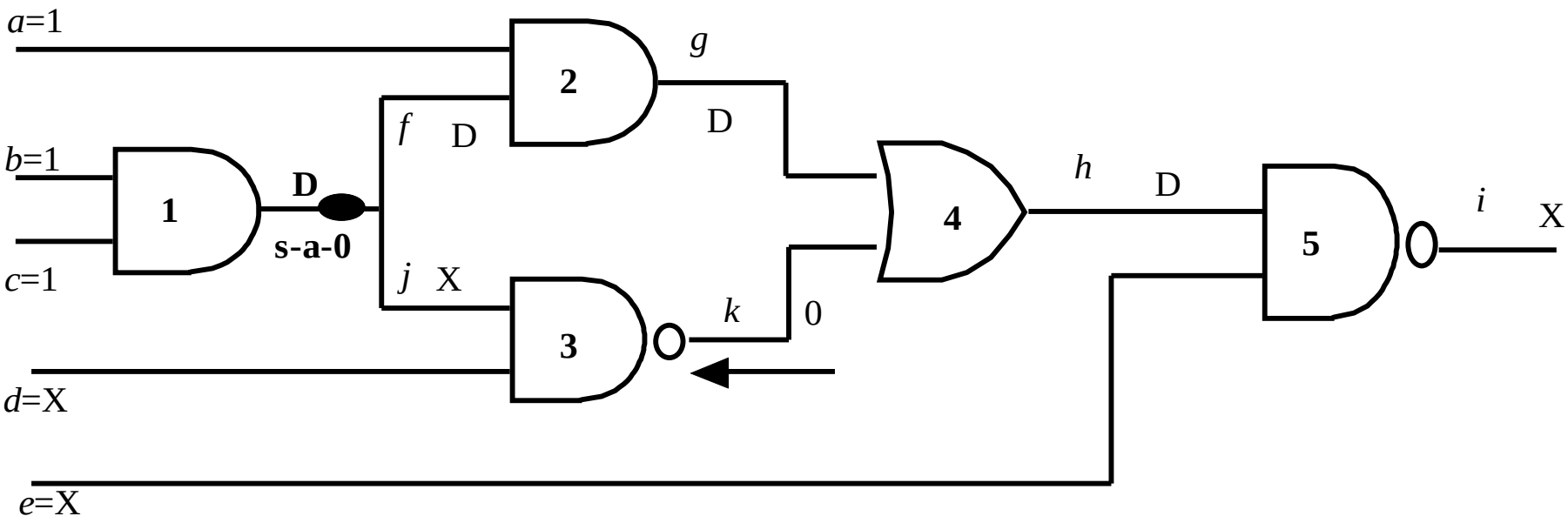
Propagation of D and implications



Propagate D through D-frontier and Backward Implication
Modified D-frontier ={4}

D-Algorithm: Example

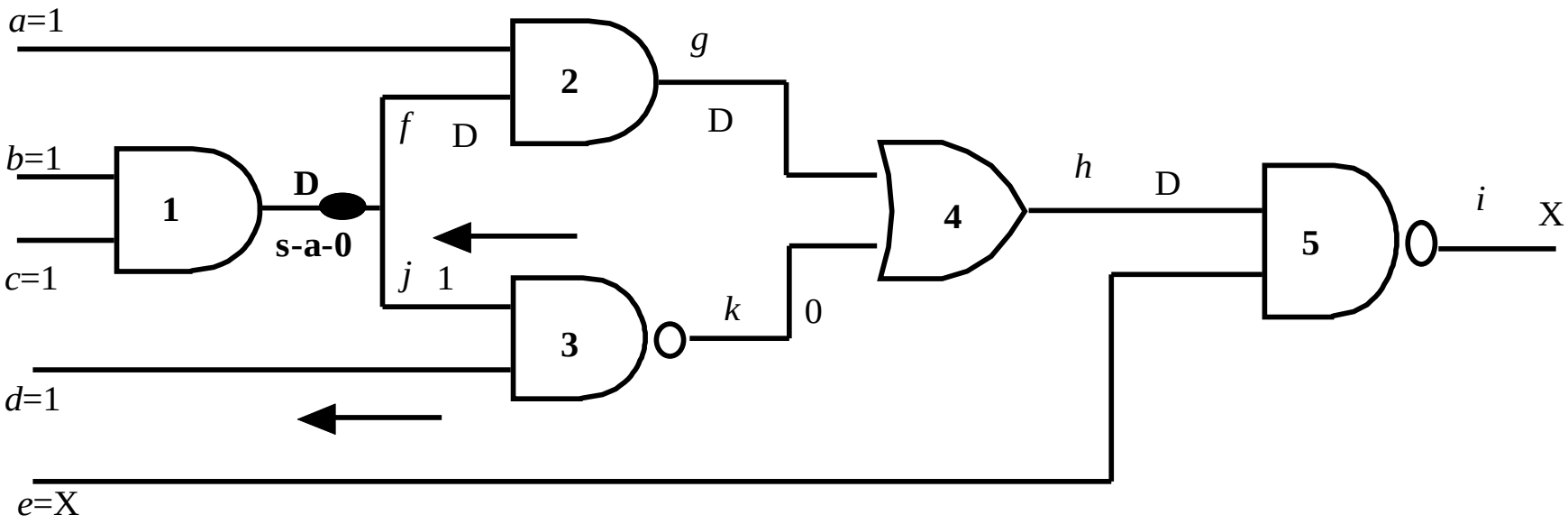
Propagation of D, implications and J-frontier



Propagate D through D-frontier and Backward Implication
Modified D-frontier ={5}
J-frontier={3}

D-Algorithm: Example

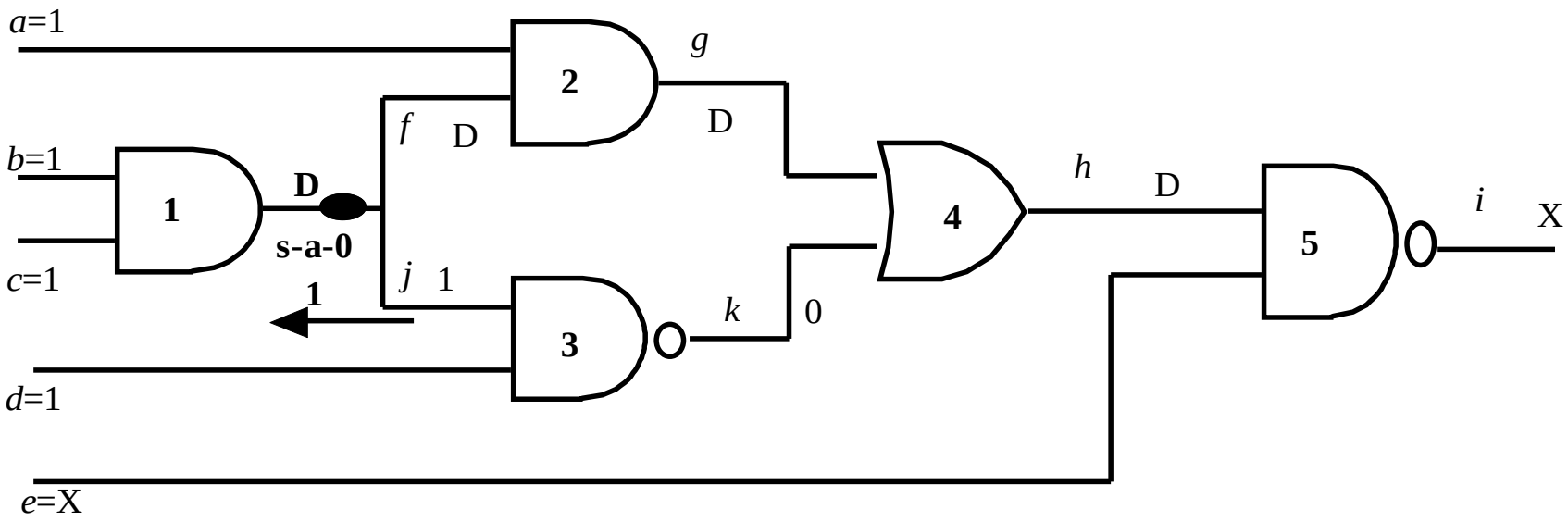
Backward implication



Backward Implication
Modified J-frontier={}

D-Algorithm: Example

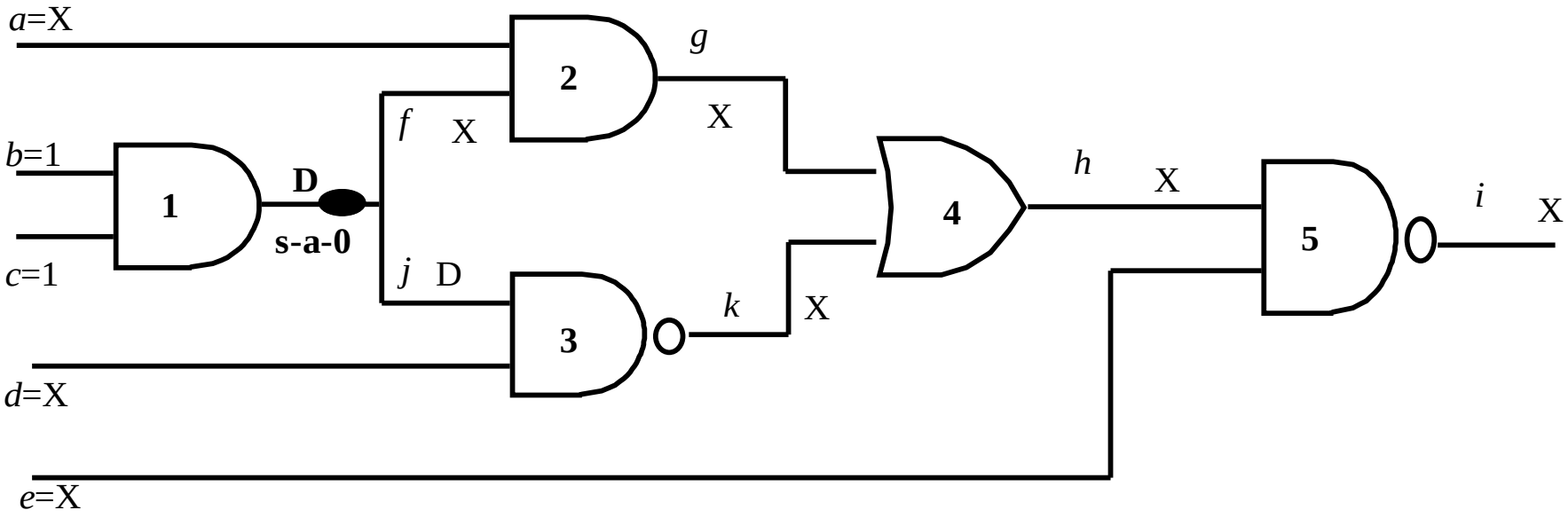
Inconsistency in net j



Backward Implication
Require output of gate 1 to be 1, which is D
So INCONSTANCY, BACKTRACK

D-Algorithm: Example

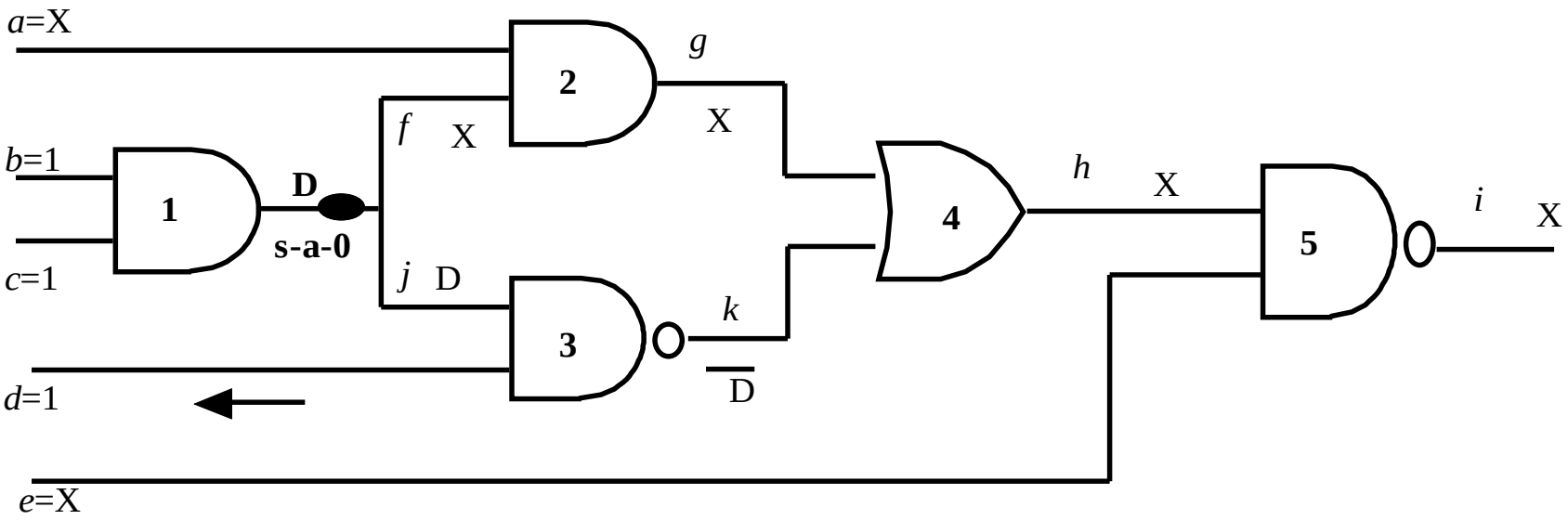
Backtrack by considering gate-3 for D propagation



Select the other gate-3 from D-frontier ={2,3}
Modified D-frontier ={3}

D-Algorithm: Example

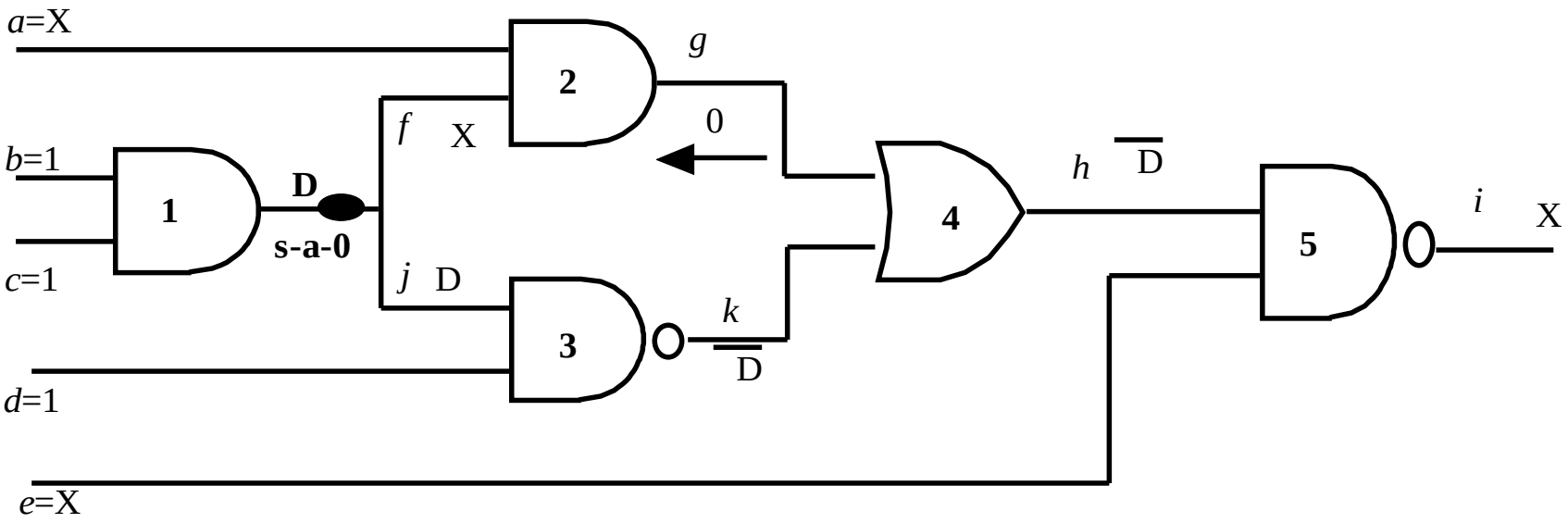
Propagation of D and implications



Propagate D through D-frontier and Backward Implication
Modified D-frontier ={4}

D-Algorithm: Example

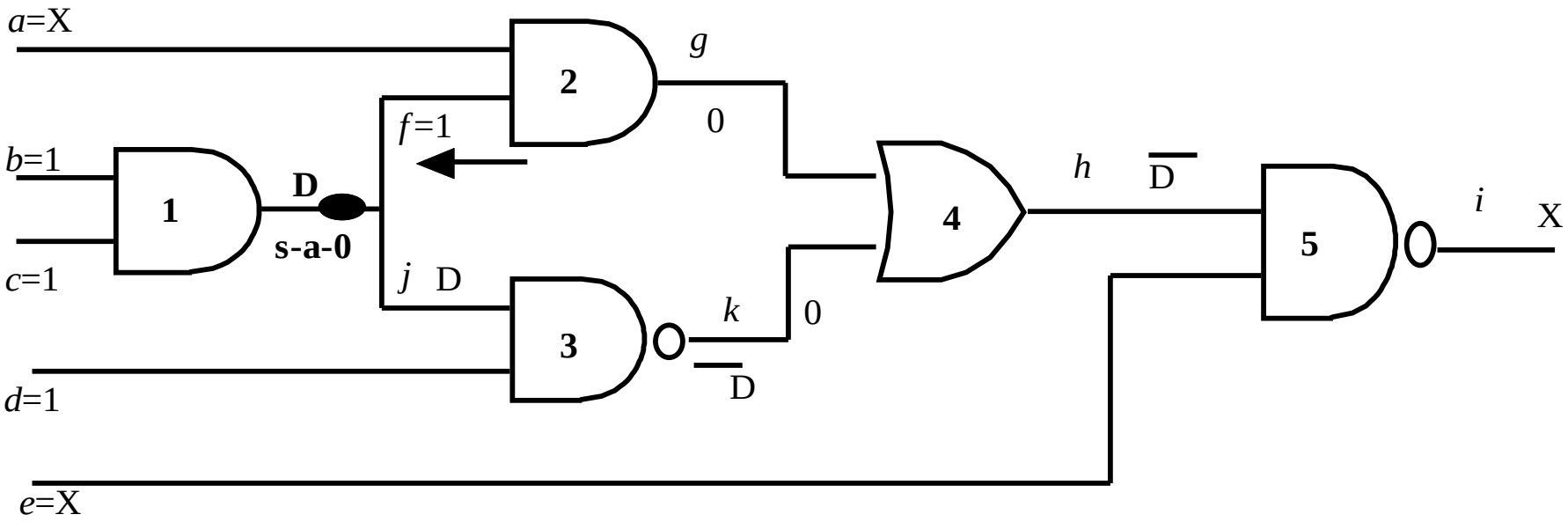
Propagation of D, implications and D and J-frontier changes



Propagate D through D-frontier and Backward Implication
Modified D-frontier ={5}
J-frontier={2}

D-Algorithm: Example

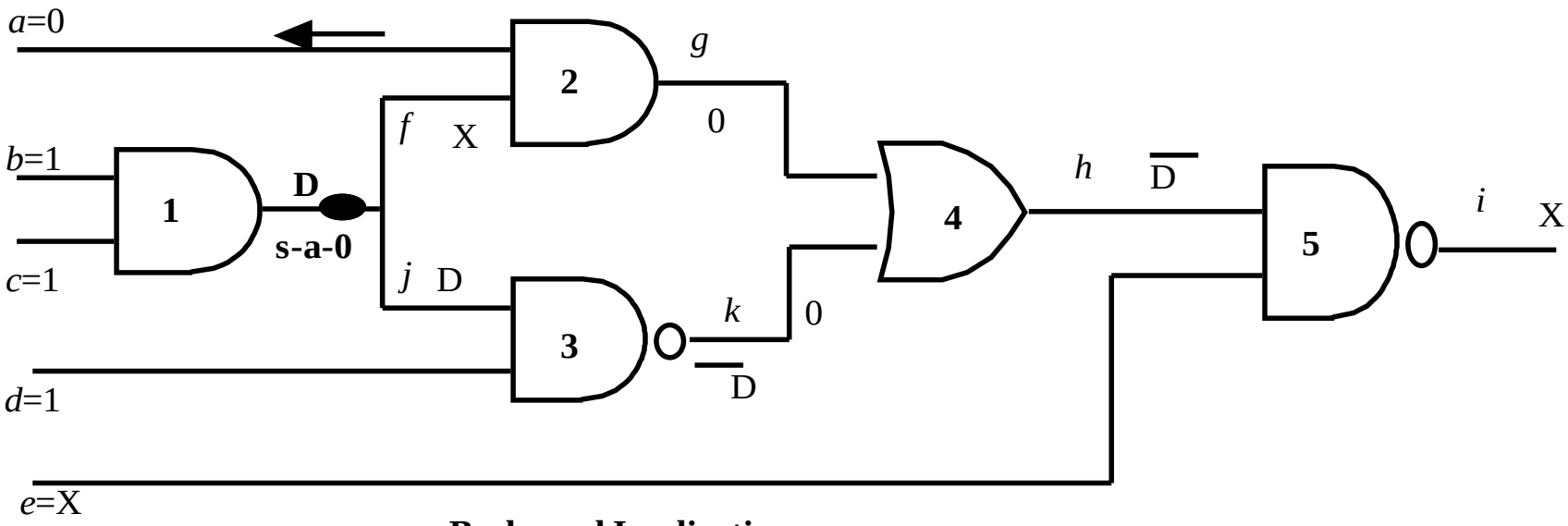
Inconsistency in net f



Backward Implication
Modified J-frontier={}
Inconsistency at net *f*

D-Algorithm: Example

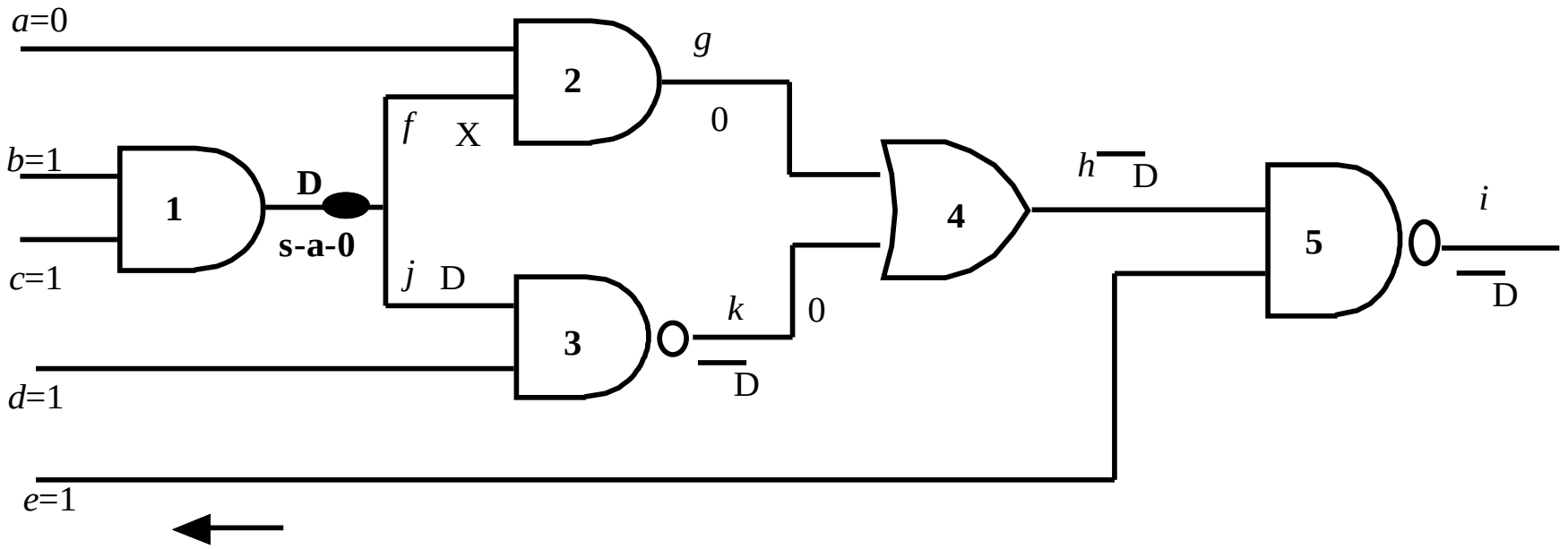
Backtrack by considering a=0, f=X for backward implication for g=0



Backward Implication
Modified J-frontier={}

D-Algorithm: Example

Backward implication for e=1 and successful test pattern generation



Propagate D through D-frontier and Backward Implication
Modified D-frontier ={}-Error propagated to PO
No inconsistency,
Test patten *a*=0,*b*=1,*c*=1,*d*=1,*e*=1,
Expected output-- if normal =1, if s-a fault =0

Why advanced ATPG algorithms: PODEM and FAN

If we observe the steps in the D-algorithm we can notice that it has three basic points

- Sensitize the fault location (i.e., D/$\overline{D}$ in the fault net)
- Propagation of D/$\overline{D}$ to a primary output, using *any* gates in D-frontier
- Use *any* values of signals as per forward and backward implications, to get the values of primary inputs, which sensitize the fault and propagate it to an output.

At any point, backtrack is required in case D-frontier is exhausted or an inconsistency is found.

Why advanced ATPG algorithms: PODEM and FAN

So it may be stated that D-algorithm provides all the steps required for ATPG, however, says nothing about which of the alternatives are to be taken for propagation of D/$\overline{D}$ or values of signals as per forward and backward implications. In other words, the algorithm does not prioritize among the options available in case of propagation of D/$\overline{D}$ or signal assignment during implications. Also, in case of backtrack, information regarding the reason for inconsistency is not used for future selection of alternatives.

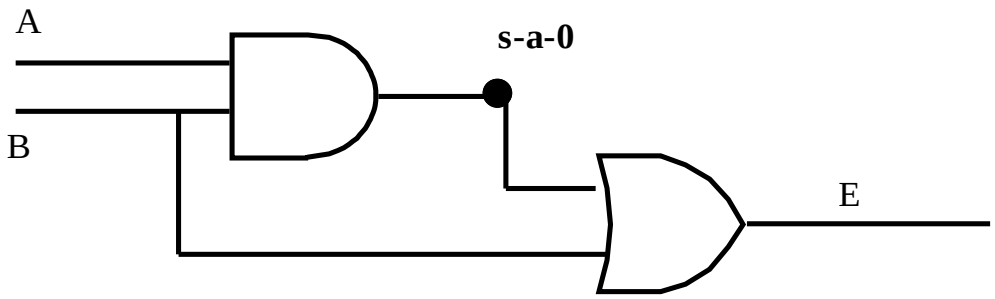
Why advanced ATPG algorithms: PODEM and FAN

So the advanced algorithms basically improve D-algorithm by providing a guided choice of alternatives based on several testability measures namely,

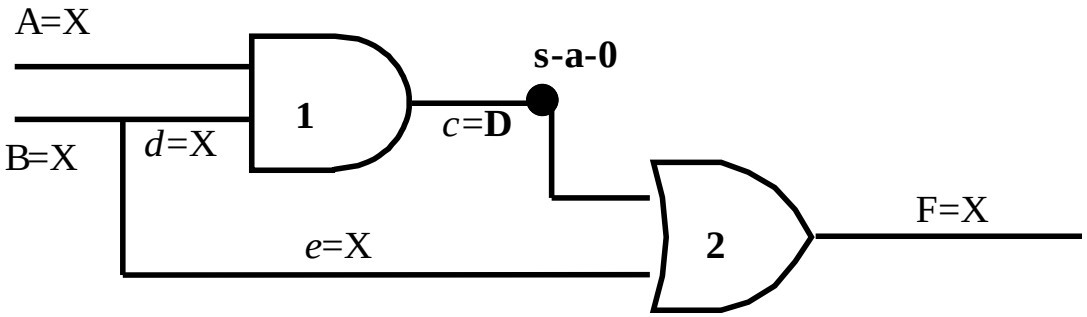
- After a fault is sensitized, if there are more than one gate in the D-frontier, then choice can be made based on conditions like,
  - The shortest path (in terms of number of gates) from the fault site to a primary output is to be taken
  - The path having gate inputs (other than the one used for fault propagation) with low values of CC1 and CC0 (SCOAP values) are preferred than the ones with higher values.
- If more than one signal values are possible for nets after implication, consider the one that will result in low CC1 and CC0 values.
- Use the reasons for inconsistency for future selection of alternatives

Questions and Answers

Question 1: Using D-algorithm show that the circuit given below is un-testable:



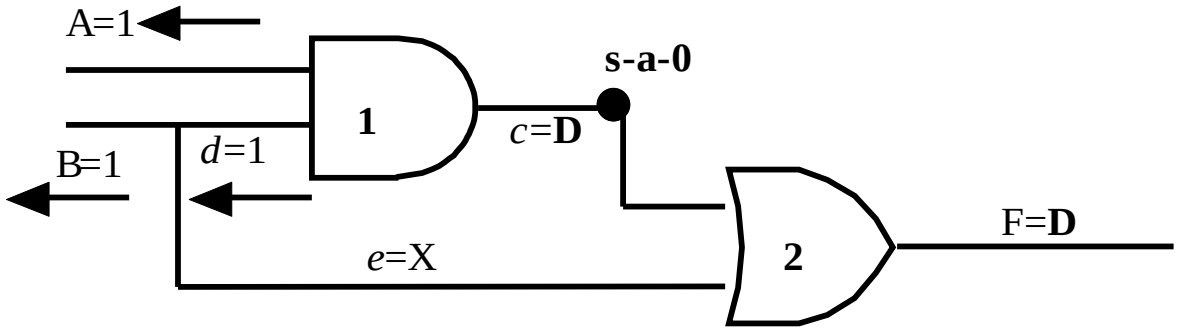
Answer



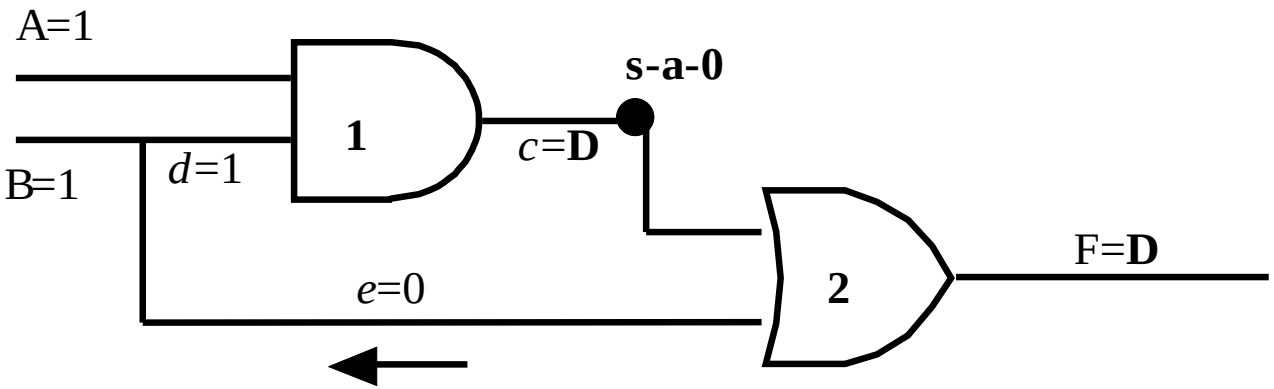
D-frontier ={2}
J-frontier ={1}

Questions and Answers

Answer



D-frontier ={}
Backward Implication



Backward Implication
Inconstancy