



Metodologie di progetto HW

Il test di circuiti digitali

Versione del 9/04/04



Metodologie di progetto HW

Il test di circuiti digitali

Combinational ATPG Basics

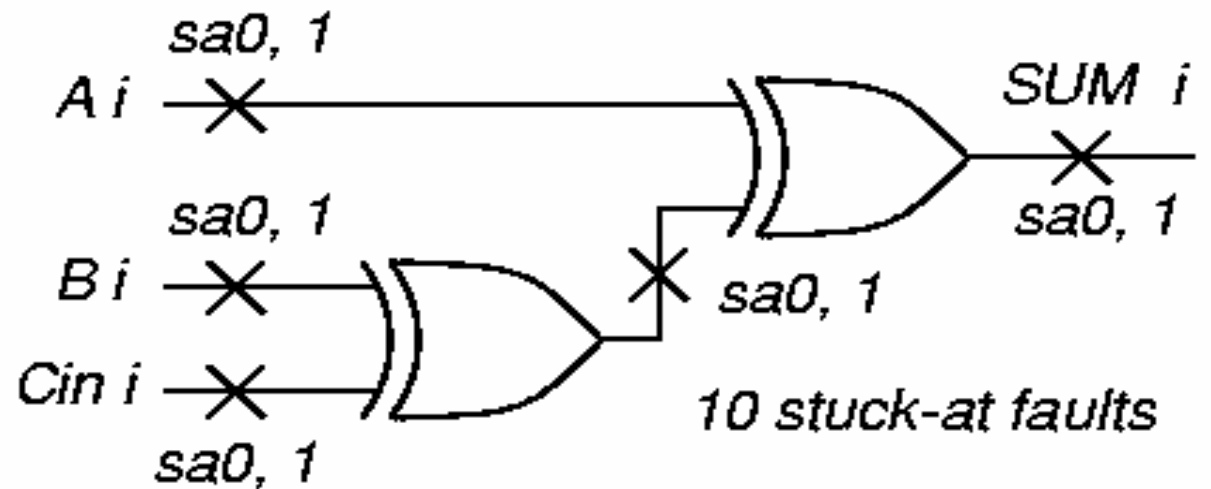
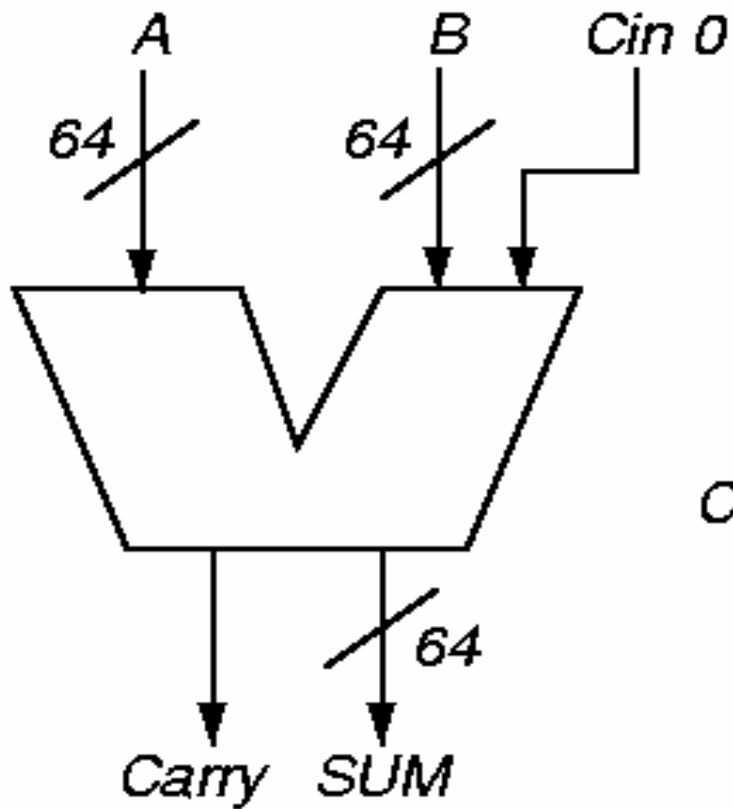


Overview

- ❑ Structural vs. functional test
- ❑ Definitions
- ❑ Completeness
- ❑ Conditions for finding a test
- ❑ Algebras
- ❑ Types of Algorithms - classical
- ❑ Complexity
- ❑ Summary

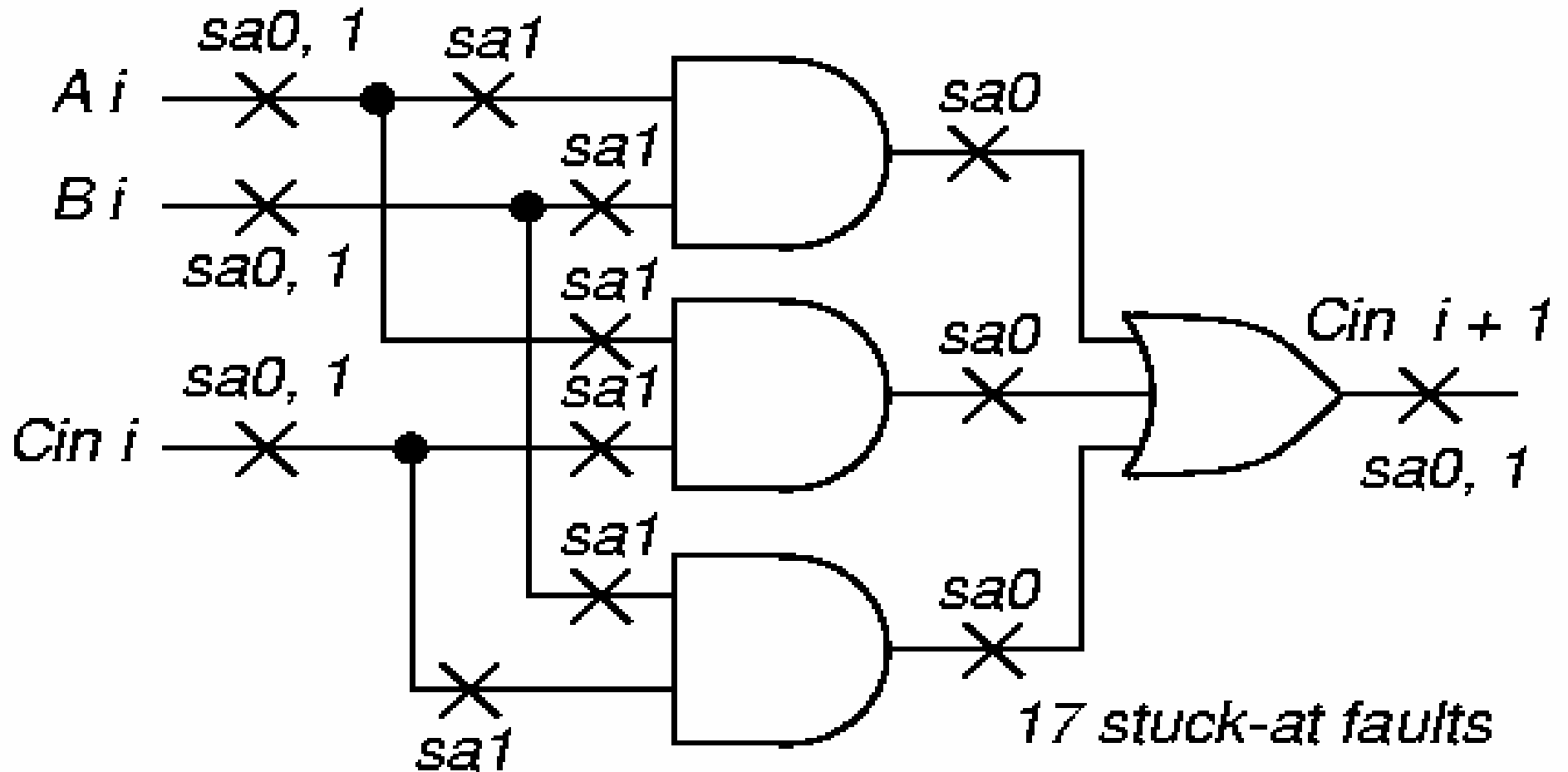


Functional vs. Structural ATPG





Carry Circuit





Functional vs. Structural (Contd.)

- **Functional ATPG** - generate complete set of tests for circuit input-output combinations
 - 129 inputs, 65 outputs:
 - $2^{129} = 680,564,733,841,876,926,926,749,$
214,863,536,422,912 patterns
 - Using 1 GHz ATE, would take 2.15×10^{22} years
- **Structural test:**
 - No redundant adder hardware, 64 bit slices
 - Each with 27 faults (using fault equivalence)
 - At most $64 \times 27 = 1728$ faults (tests)
 - Takes 0.000001728 s on 1 GHz ATE
- **Designer gives small set of functional tests** - augment with structural tests to boost coverage to 98+ %



Definition of Automatic Test-Pattern Generator

- **Standard ATPG approach:**
 - Inject fault into circuit modeled in computer
 - Use various ways to activate and propagate fault effect through hardware to circuit output
 - Output flips from expected to faulty signal
- ***Electron-beam (E-beam) test*** observes internal signals
 - “picture” of nodes charged to 0 and 1 in different colors
 - Too expensive
- ***Scan design*** - add test hardware to all flip-flops to make them a giant shift register in test mode
 - Can shift state in, scan state out
 - Widely used - makes sequential test combinational
 - Costs: 5 to 20% chip area, circuit delay, extra pin, longer test sequence



Algorithm Completeness

- ❑ **Definition:** Algorithm is *complete* if it ultimately can search entire binary (decision) space, as needed, to generate a test
- ❑ *Untestable fault* - no test for it even after entire space is searched
- ❑ Combinational circuits **only** - untestable faults are *redundant*, showing the presence of unnecessary hardware



Notation

Symbol	Meaning	Good Machine	Failing Machine	
D	1/0	1	0	Roth's Algebra
\overline{D}	0/1	0	1	
0	0/0	0	0	
1	1/1	1	1	
X	X/X	X	X	
G0	0/X	0	X	Muth's Additions
G1	1/X	1	X	
F0	X/0	X	0	
F1	X/1	X	1	



Roth's and Muth's Higher-Order Algebras

- Represent two machines, which are simulated simultaneously by a computer program:
 - Good circuit machine (1st value)
 - Bad circuit machine (2nd value)
- Better to represent both in the algebra:
 - Need only 1 pass of ATPG to solve both
 - Good machine values that preclude bad machine values become obvious sooner & vice versa
- Needed for complete ATPG:
 - Combinational: Multi-path sensitization, Roth Algebra
 - Sequential: Muth Algebra -- good and bad machines may have different initial values due to fault



Conditions for Finding a Test

- Fault excitation
 - the signal value at the fault site must be different from the value of the stuck-at fault (thus fault site must contain a D or a \overline{D})
- The fault effect must be **propagated to a primary output** (a D or a \overline{D} must appear at the output)
- Some simple **observations**
 - There must be at least a D or a \overline{D} on some circuit nets
 - D's must form a chain to some output



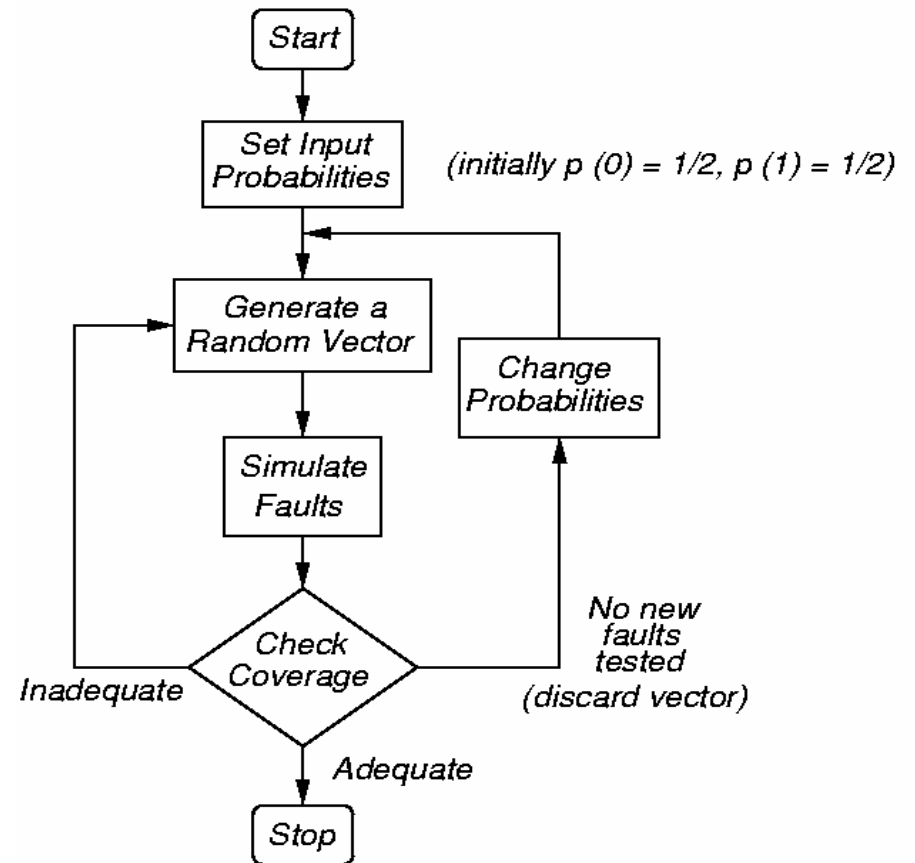
Exhaustive Algorithm

- For n -input circuit, generate all 2^n input patterns
- **Infeasible**, unless circuit is partitioned into **cones** of logic, with ≤ 15 inputs
 - Perform **exhaustive** ATPG for each cone
 - **Misses** faults that require specific activation patterns for **multiple** cones to be tested



Random-Pattern Generation

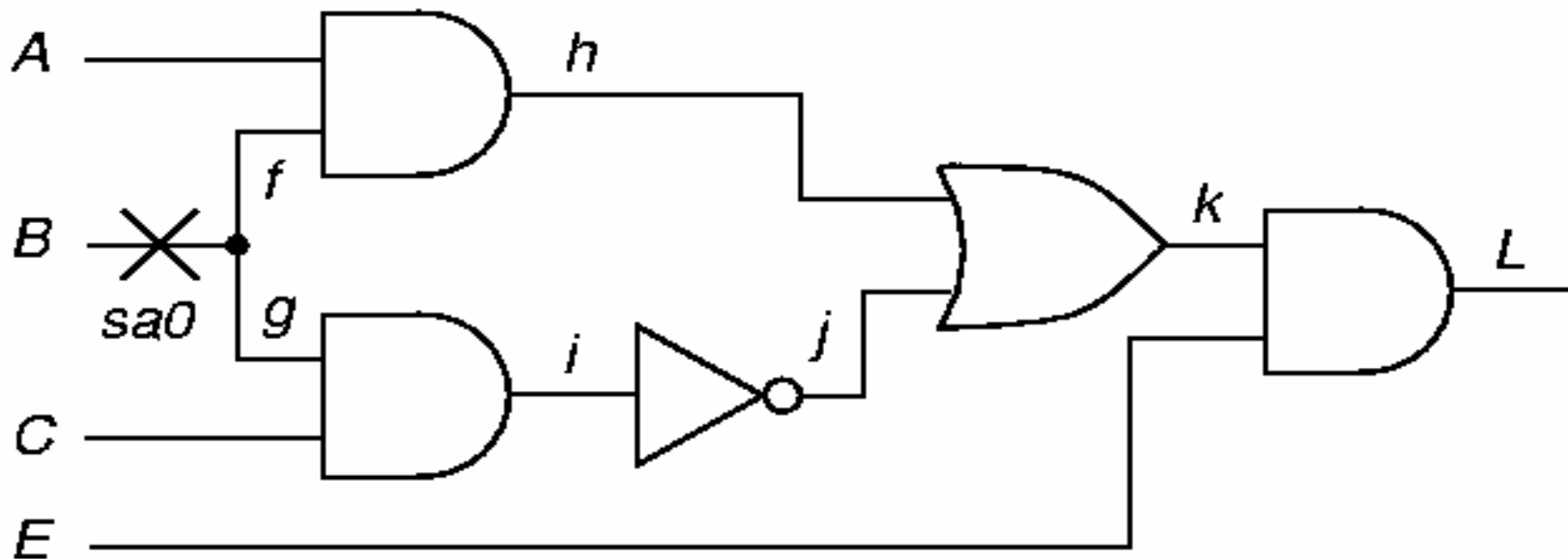
- Use to get tests for 60-80% of faults, then switch to deterministic pattern generator for rest





Path Sensitization Method - Example

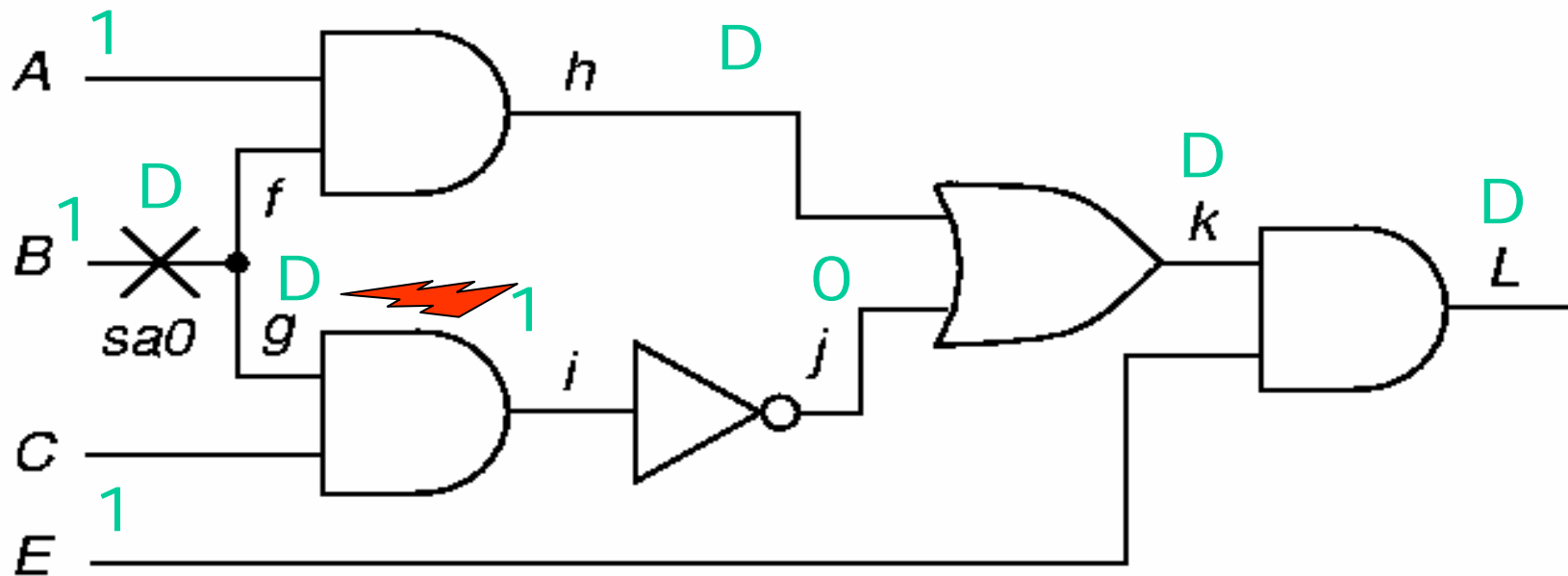
1. Fault Sensitization
2. Fault Propagation
3. Line Justification





Path Sensitization Method -Example

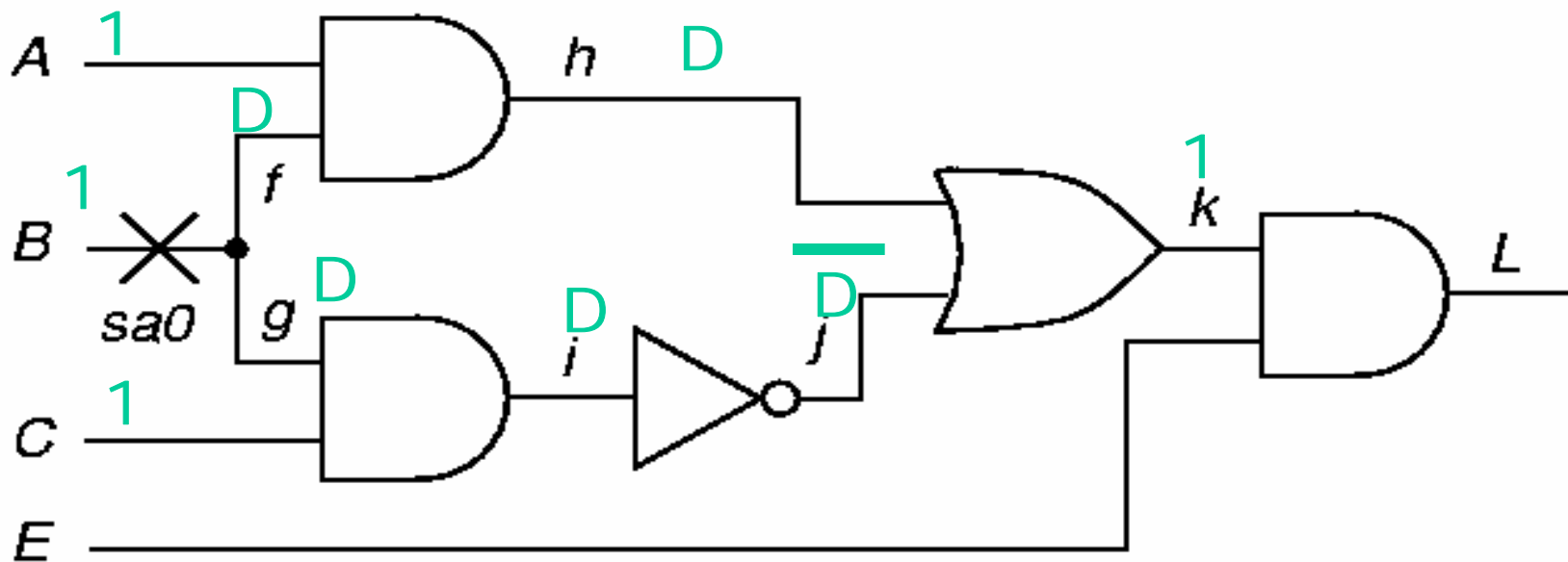
- Try path f - h - k - L. This path is blocked at j, since there is no way to justify the 1 on i





Path Sensitization Method

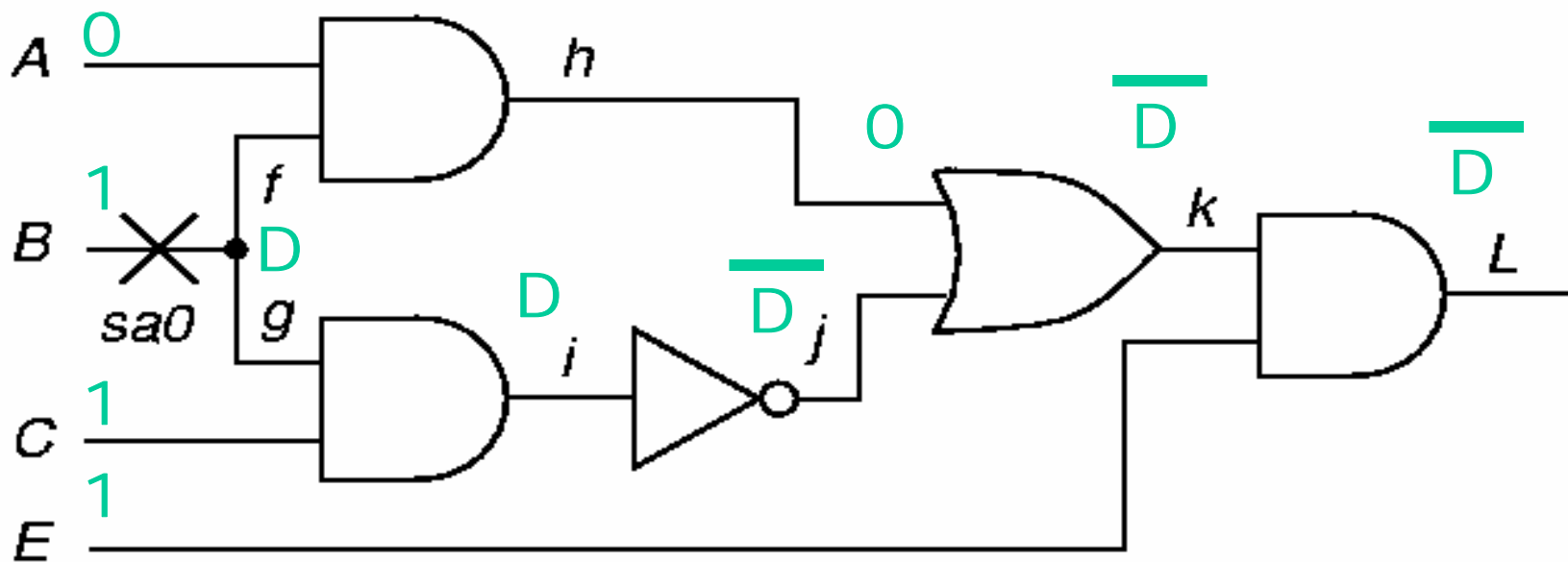
- Try simultaneous paths $f - h - k - L$ and $g - i - j - k - L$. These paths blocked at k because D -frontier (chain of D or \overline{D}) disappears





Path Sensitization Method Circuit Example

- Final try: path $g - i - j - k - L$ - test found!





Computational Complexity

- Ibarra and Sahni analysis - *NP-Complete*
 - no polynomial expression found for compute time, presumed to be exponential
- Worst case:
 - no_pi inputs, 2^{no_pi} input combinations
 - no_ff flip-flops, 4^{no_ff} initial flip-flop states
(good machine 0 or 1 ~~X~~ bad machine 0 or 1)
 - work to forward or reverse *simulate* n logic gates αn
- **Complexity:** $O(n \times 2^{no_pi} \times 4^{no_ff})$



History of Algorithm Speedups

Algorithm	Est. speedup over D-ALG (normalized to D-ALG time)	Year
D-ALG	1	1966
PODEM	7	1981
FAN	23	1983
TOPS	292	1987
SOCRATES	1574 † ATPG System	1988
Waicukauski et al.	2189 † ATPG System	1990
EST	8765 † ATPG System	1991
TRAN	3005 † ATPG System	1993
Recursive learning	485	1995
Tafertshofer et al.	25057	1997



Summary

- ❑ Basic **definitions** explained
- ❑ Developed **notation** and required algebra that will be used for test generation and fault simulation
- ❑ Basics of **test generation** developed
- ❑ **Complexity** of test generation addressed



Metodologie di progetto HW

Il test di circuiti digitali

Combinational ATPG

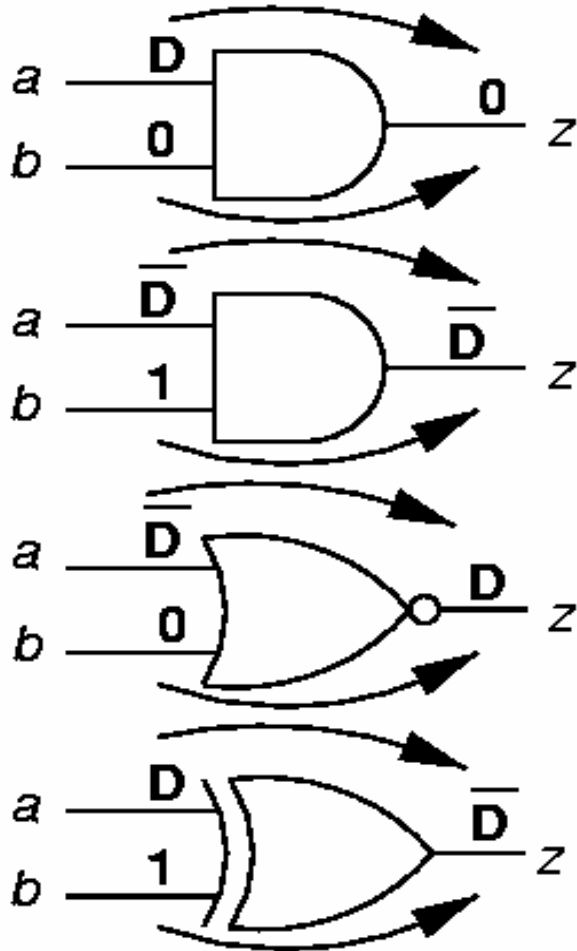


Overview: Major ATPG algorithms

- Definitions
- **D-Algorithm** (Roth) -- 1966
 - *D-cubes*
 - Bridging faults
 - Logic gate function change faults
- **PODEM** (Goel) -- 1981
 - *X-Path-Check*
 - *Backtracing*
- Summary



Forward Implication



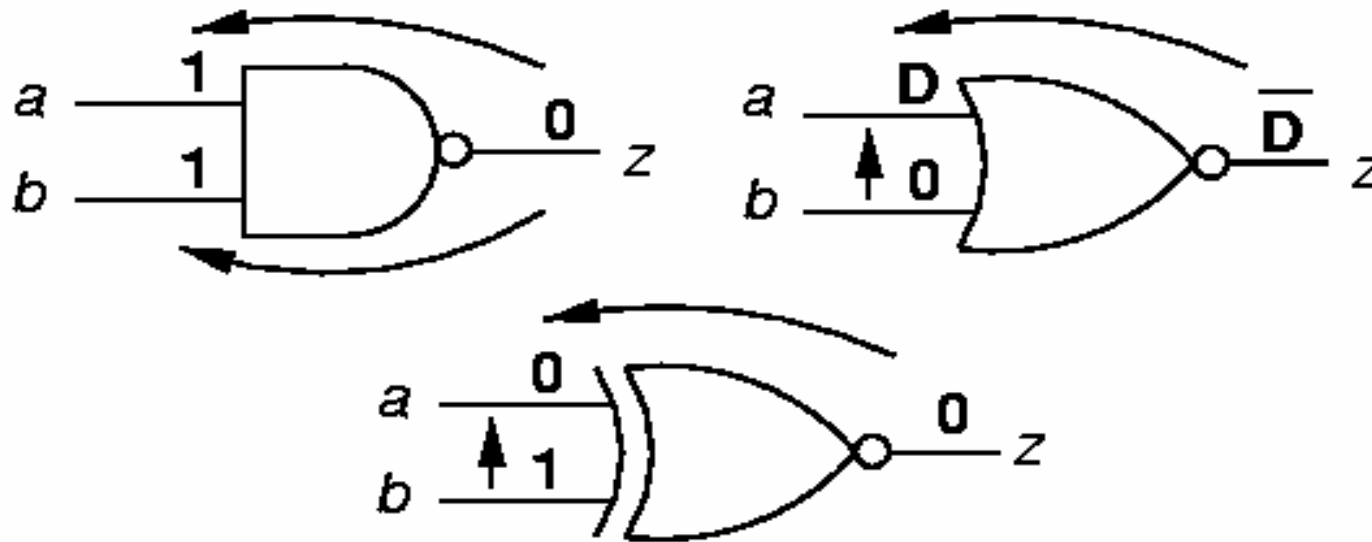
- Results in logic gate inputs that are significantly labeled so that output is uniquely determined
- AND gate forward implication table:

$a \backslash b$	0	1	X	D	\overline{D}
0	0	0	0	0	0
1	0	1	X	D	\overline{D}
X	0	X	X	X	X
D	0	D	X	D	0
\overline{D}	0	\overline{D}	X	0	\overline{D}



Backward Implication

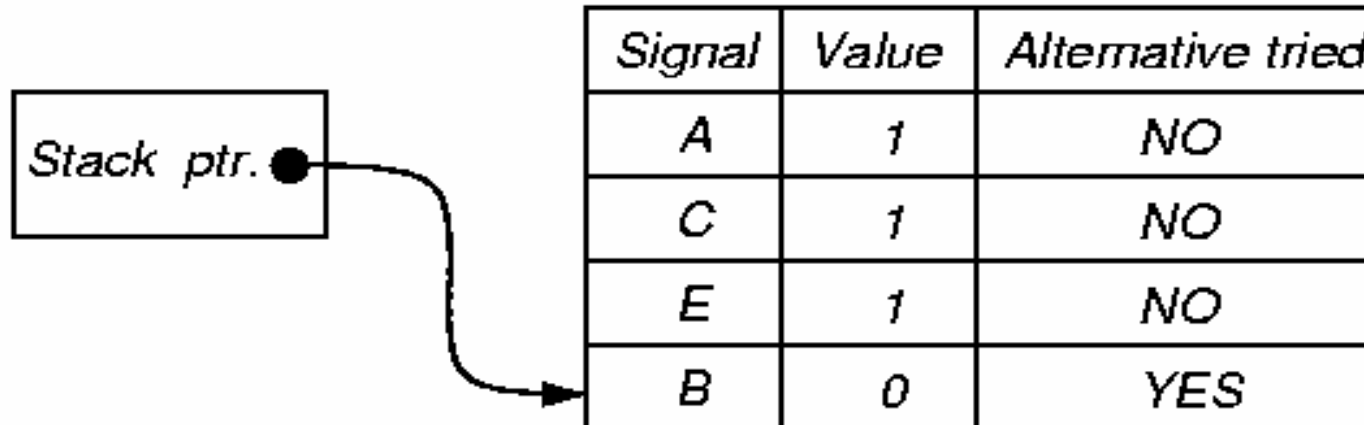
- Unique determination of all gate inputs when the gate output and some of the inputs are given





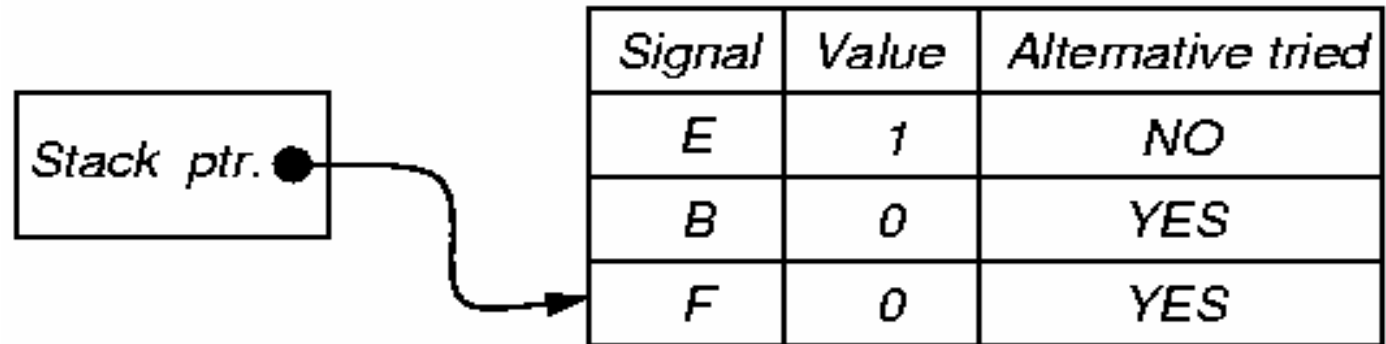
Implication Stack

- Push-down stack. Records:
 - Each signal set in circuit by ATPG
 - Whether alternate signal value already tried
 - Portion of binary search tree already searched

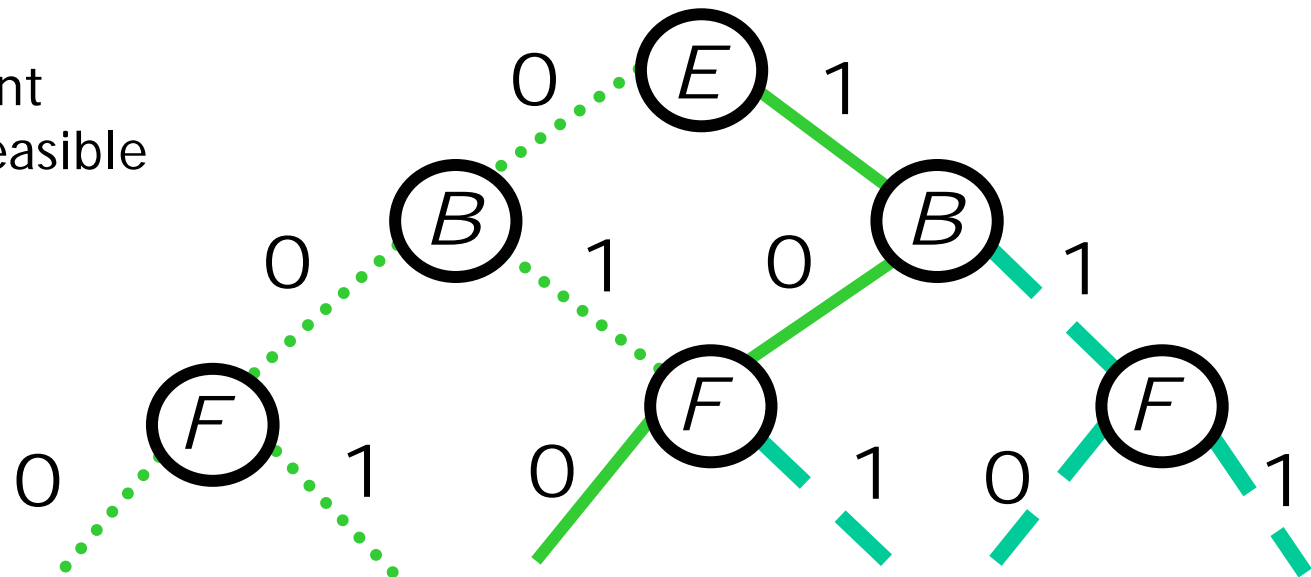




Implication Stack, Decision Tree, and Backtrack



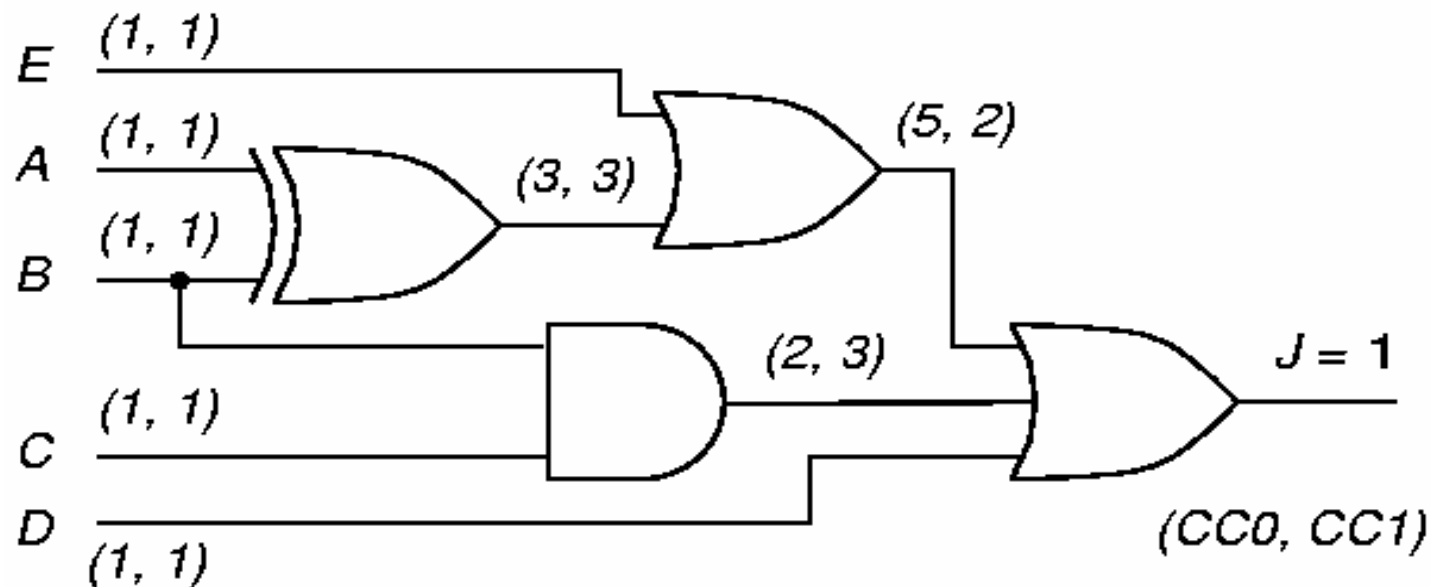
- Unexplored
- Present Assignment
- Searched and Infeasible





Objectives and Backtracing in ATPG

- *Objective* - desired signal value goal for ATPG
 - Guides it away from infeasible/hard solutions
 - Uses heuristics
- *Backtrace* - Determines which primary input and value to set to achieve objective
 - Use testability measures





Objectives and Backtracing in ATPG

- ❑ *Objective* - desired signal value goal for ATPG
 - Guides it away from infeasible/hard solutions
 - Uses heuristics
- ❑ *Backtrace* - Determines which primary input and value to set to achieve objective
 - Use heuristics such as nearest PI
- ❑ *Forward trace* - Determines gate through which the fault effect should be sensitized
 - Use heuristics such as output that is closest to the present fault effect



Branch-and-Bound Search

- **Efficiently** searches binary search tree
- **Branching** - At each tree level, selects which input variable to set to what value
- **Bounding** - Avoids exploring large tree portions by artificially restricting search decision choices
 - Complete exploration is impractical
 - Uses heuristics



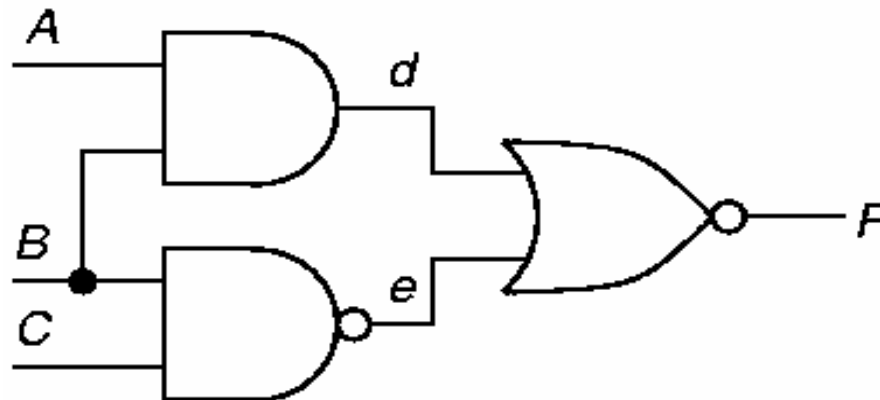
D-Algorithm - Roth (1966)

- Fundamental concepts invented:
 - First complete ATPG algorithm
 - *D-Cube*
 - *D-Calculus*
 - *Implications* - forward and backward
 - *Implication stack*
 - *Backtrack*
 - Test Search Space



Singular Cover Example

- Minimal set of logic signal assignments to represent a function
 - show prime implicants and prime implicates of Karnaugh map (with explicitly showing the outputs too)



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 $\overline{\overline{D}}$'s to D's (do both)
- ❑ AND gate:

	<i>A</i>	<i>B</i>	<i>d</i>
Rows 1 & 3	D	1	D
Reverse inputs	1	D	D
And two cubes	\overline{D}	\overline{D}	\overline{D}
Interchange D and \overline{D}	D	\overline{D}	\overline{D}
	1	D	D
	\overline{D}	1	\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*: $0 \cap 0 = 0 \cap X = X \cap 0 = 0$
 $1 \cap 1 = 1 \cap X = X \cap 1 = 1$
 $X \cap X = X$

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

\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}
\bar{D}	ψ	ψ	\bar{D}	μ	λ
D	ψ	ψ	D	λ	μ



Primitive D-Cube of Failure

- Models circuit faults:
 - *Stuck-at-0*
 - *Stuck-at-1*
 - Other faults, such as *Bridging fault* (short circuit)
 - Arbitrary change in logic function
- AND Output sa0: " 1 1 \overline{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

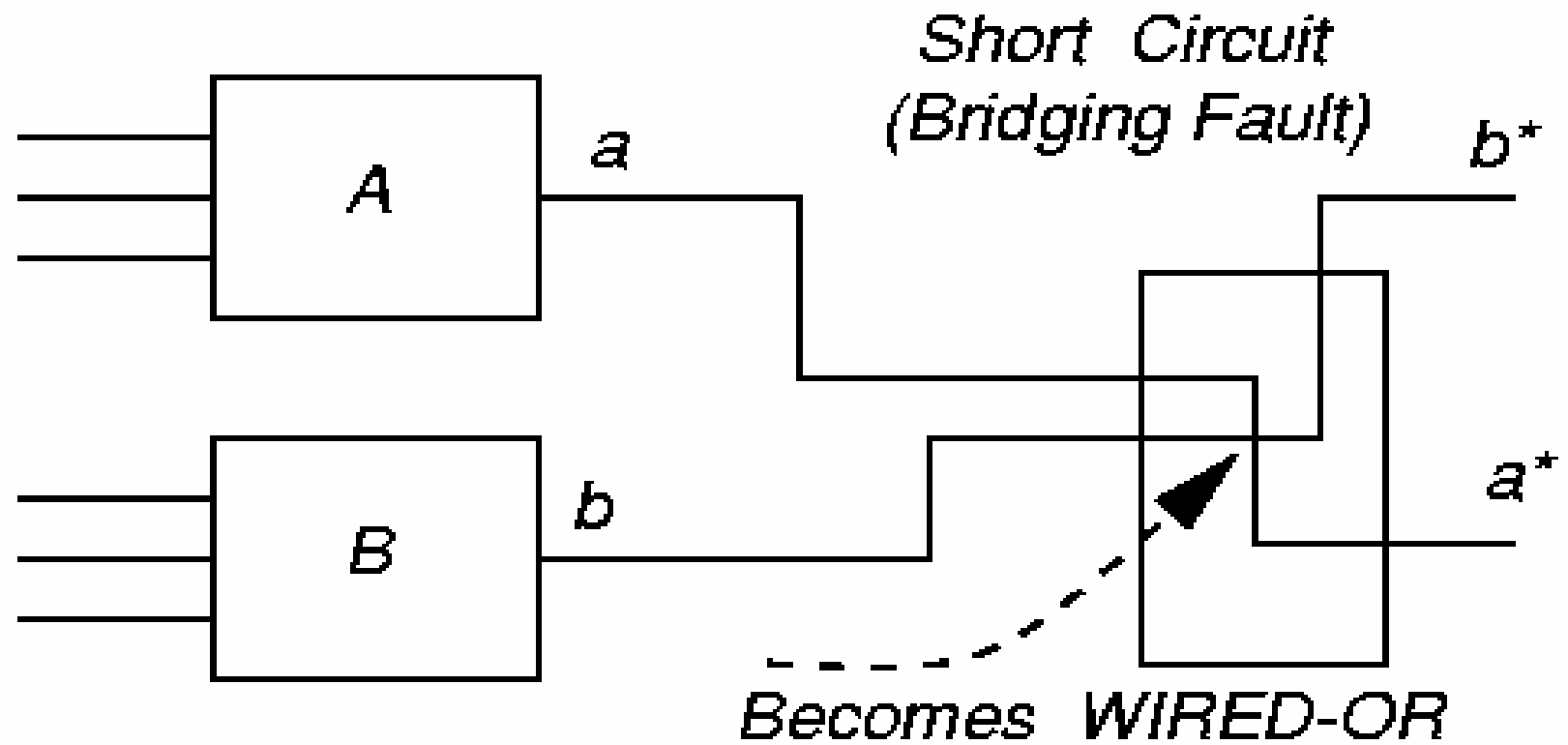


Implication Procedure

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



Bridging Fault Circuit





Construction of Primitive D-Cubes of Failure

1. Make cube set α_1 when good machine output is 1 and set α_0 when good machine output is 0
2. Make cube set β_1 when failing machine output is 1 and β_0 when it is 0
3. Change α_1 outputs to 0 and D-intersect each cube with every β_0 . If intersection works, change output of cube to D
4. Change α_0 outputs to 1 and D-intersect each cube with every β_1 . If intersection works, change output of cube to \bar{D}

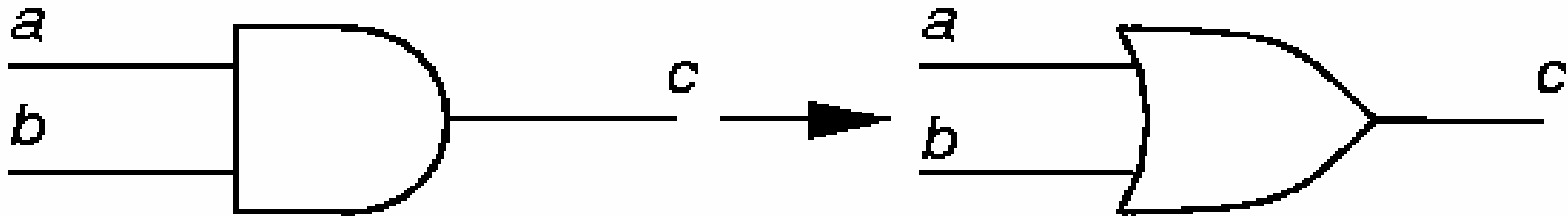


Bridging Fault D-Cubes of Failure

Cube-set	a	b	a^*	b^*	Cube-set	b	a^*	b^*	a
$\alpha 0$	0	X	0	X	PDFs for Bridging fault	0	$\overline{1}$	\overline{D}	1
	X	0	X	0		1	\overline{D}	1	0
$\alpha 1$	1	X	1	X					
	X	1	X	1					
$\beta 0$	0	0	0	0					
$\beta 1$	X	1	1	1					
	1	X	1	1					



Gate Function Change D-Cube of Failure



Cube-set	<i>a</i>	<i>b</i>	<i>c</i>	Cube-set	<i>a</i>	<i>b</i>	<i>c</i>	
$\alpha 0$	0	X	0	PDFs for AND changing to OR	0	1	\overline{D}	
	X	0	0				\overline{D}	
$\alpha 1$	1	1	1			1	0	\overline{D}
$\beta 0$	0	0	0					\overline{D}
$\beta 1$	1	X	1		1		0	\overline{D}
	X	1	1					\overline{D}



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 (\bar{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  $\bar{D}$  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 \bar{g} , not a PI;
 if (inputs to gate z are both D and 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;

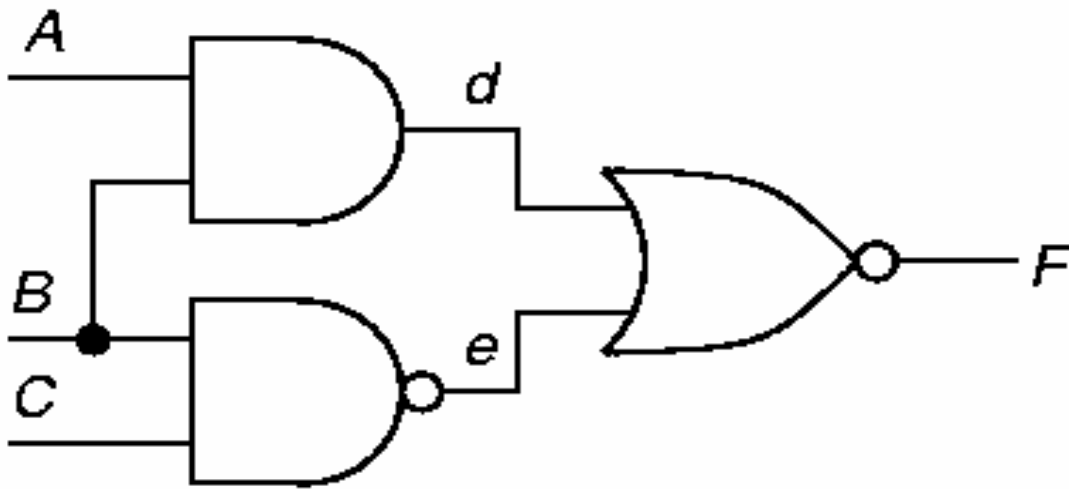


Backtrack

```
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;
```



Circuit Example 7.1 and Truth Table



Inputs			Output
<i>a</i>	<i>b</i>	<i>c</i>	<i>F</i>
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 & Propagation D-Cubes

A	B	C	d	e	F
1 0	1 0 1 0	 1 0	1 0 0 1	 0 1 1 1 0	 0 0 1
D 1 D	1 D D D D 1 D	 1 D D	D D D D D D D D D D	 	

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



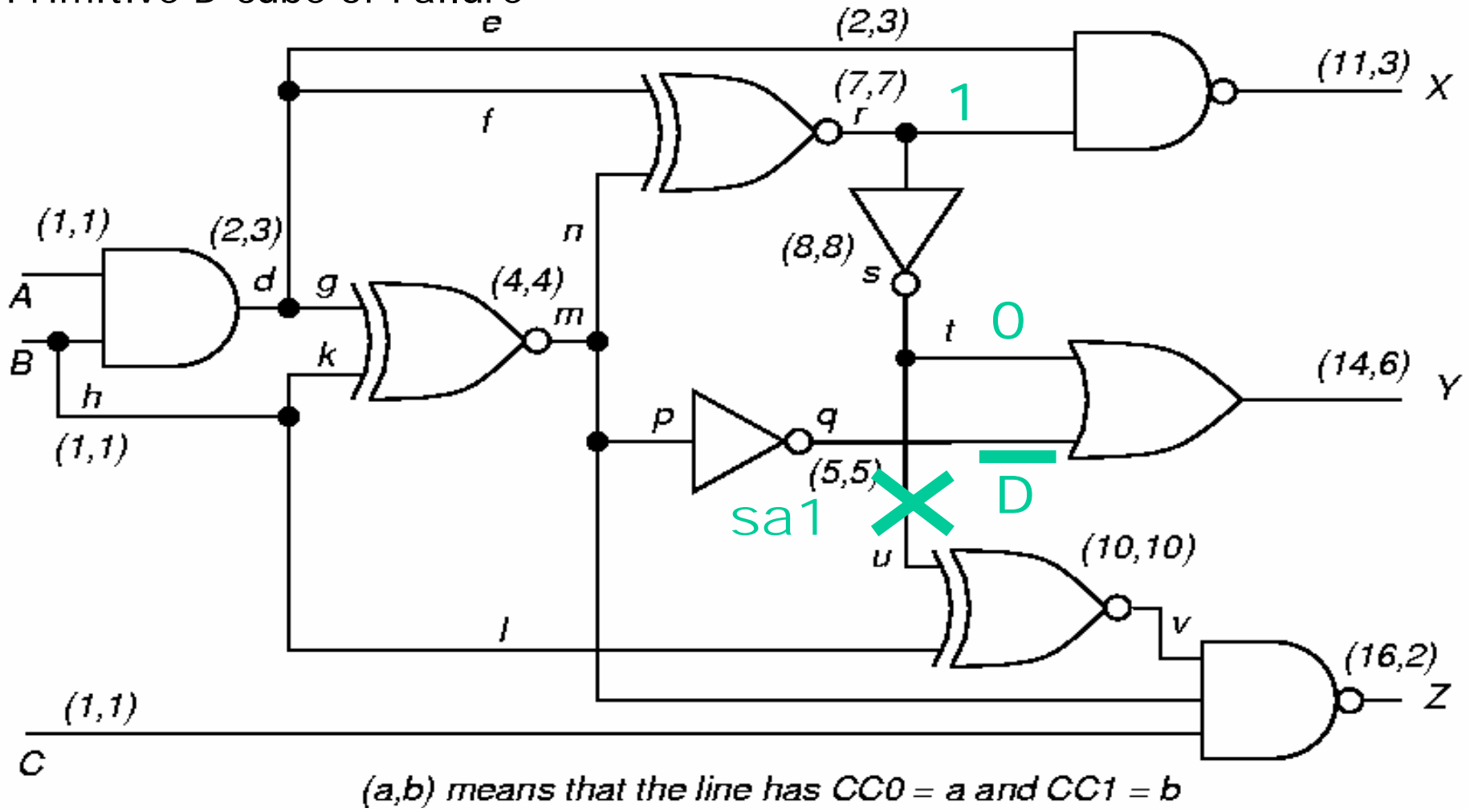
Steps for Fault d sa0

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



Example 7.3 - Fault u sa1

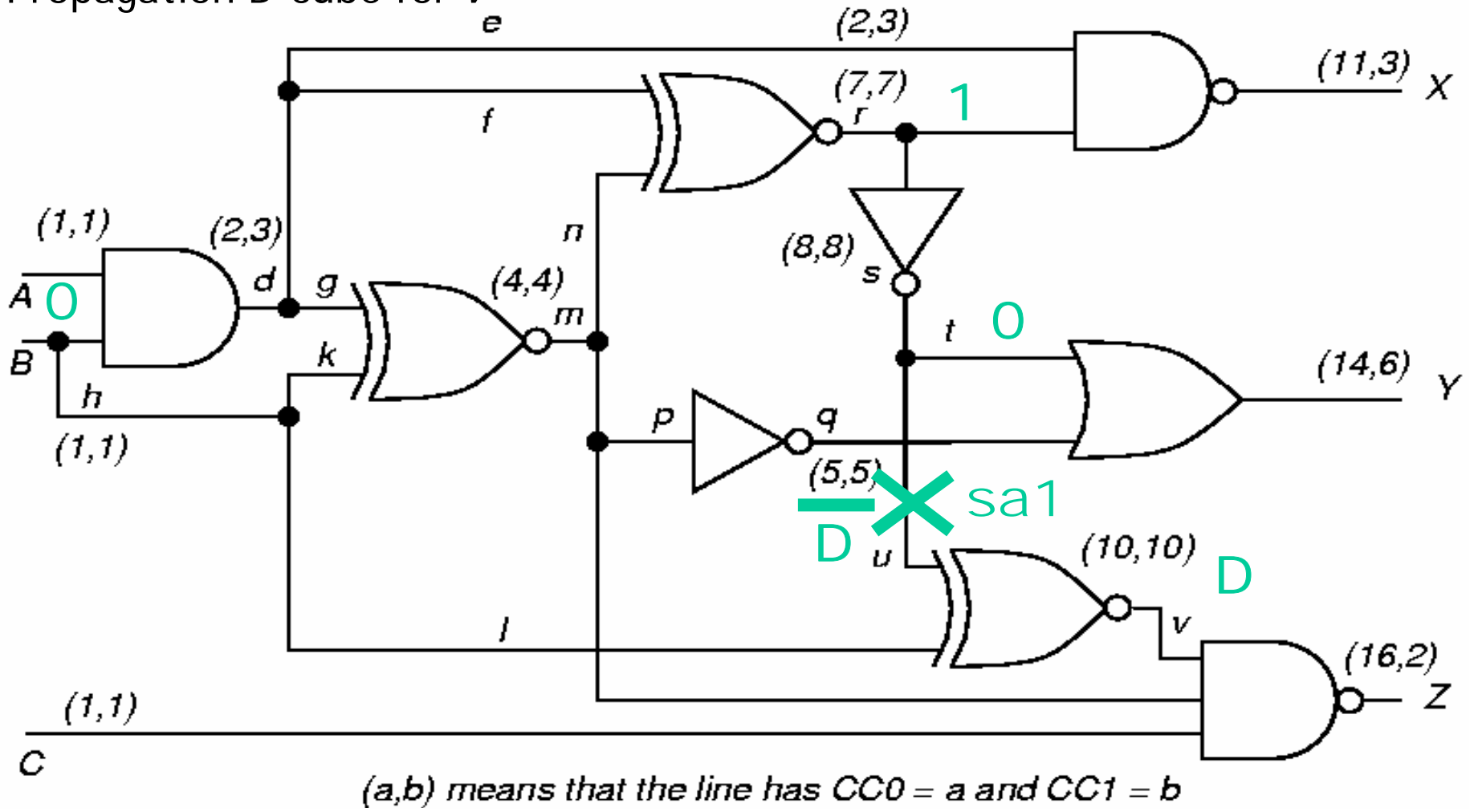
- Primitive D-cube of Failure





Example 7.3 - Step 2 $u sa1$

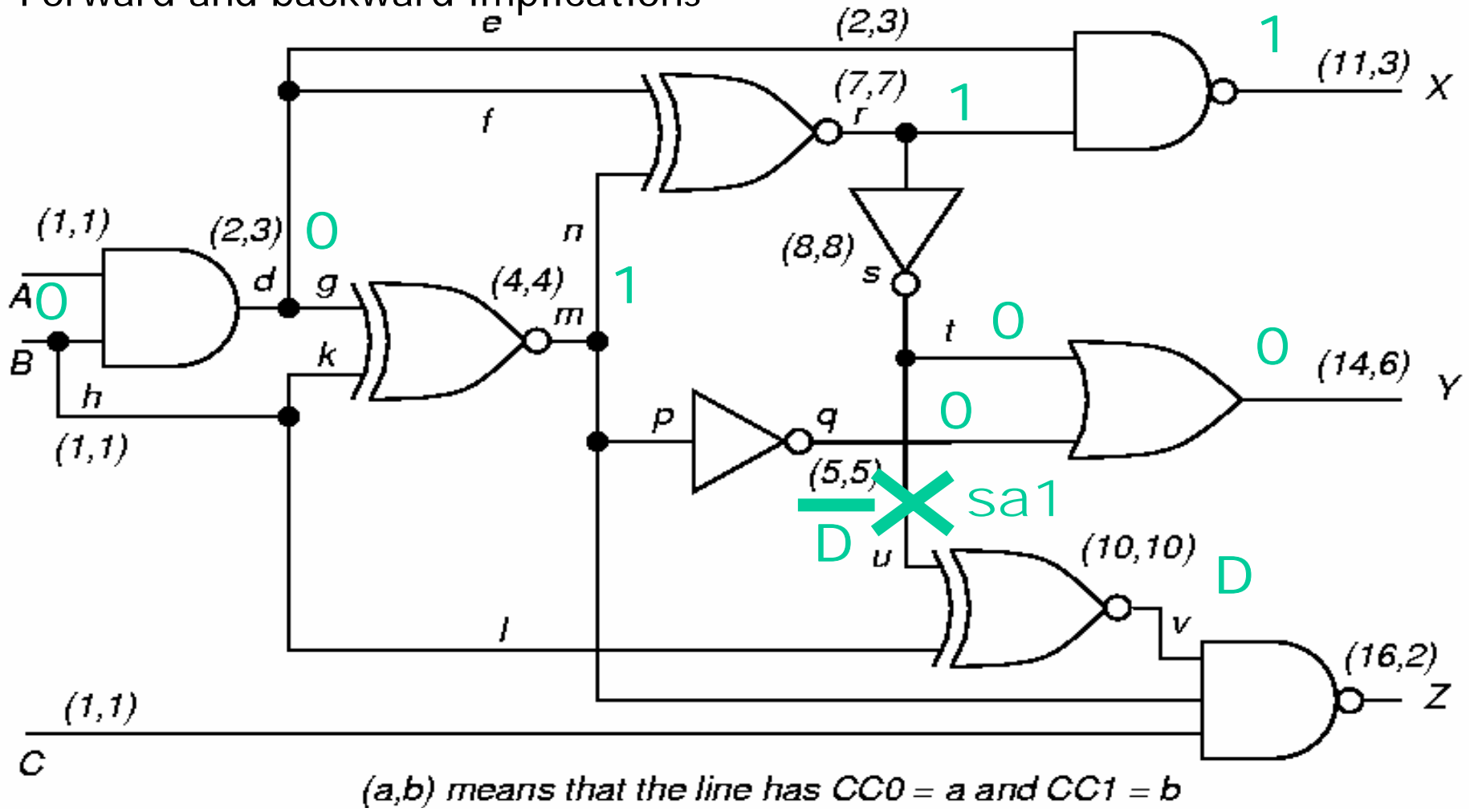
- Propagation D-cube for v





Example 7.3 - Step 2 *u sa1*

- Forward and backward implications





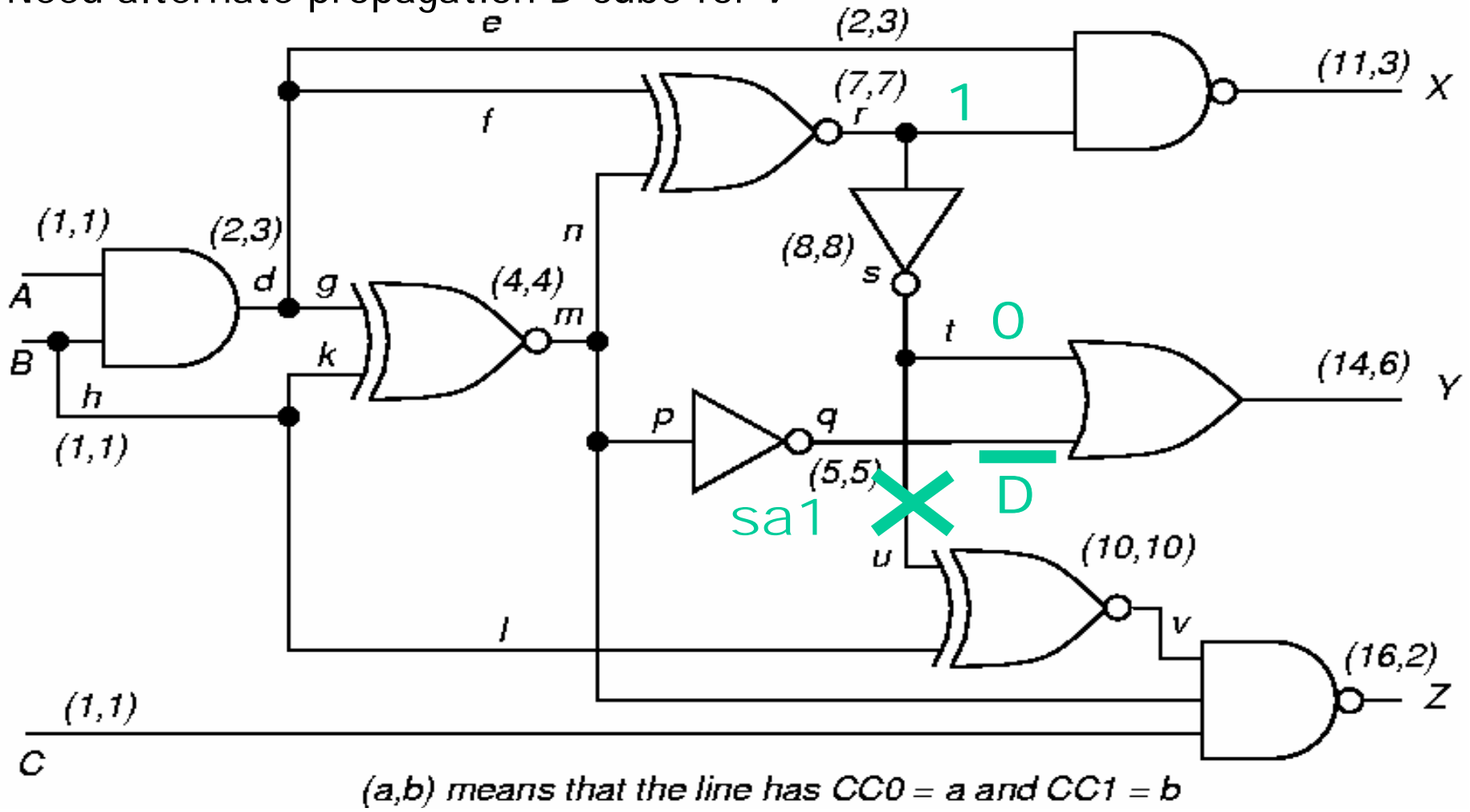
Inconsistent

- $d = 0$ and $m = 1$ cannot justify $r = 1$ (equivalence)
 - Backtrack
 - Remove $B = 0$ assignment



Example 7.3 - Backtrack

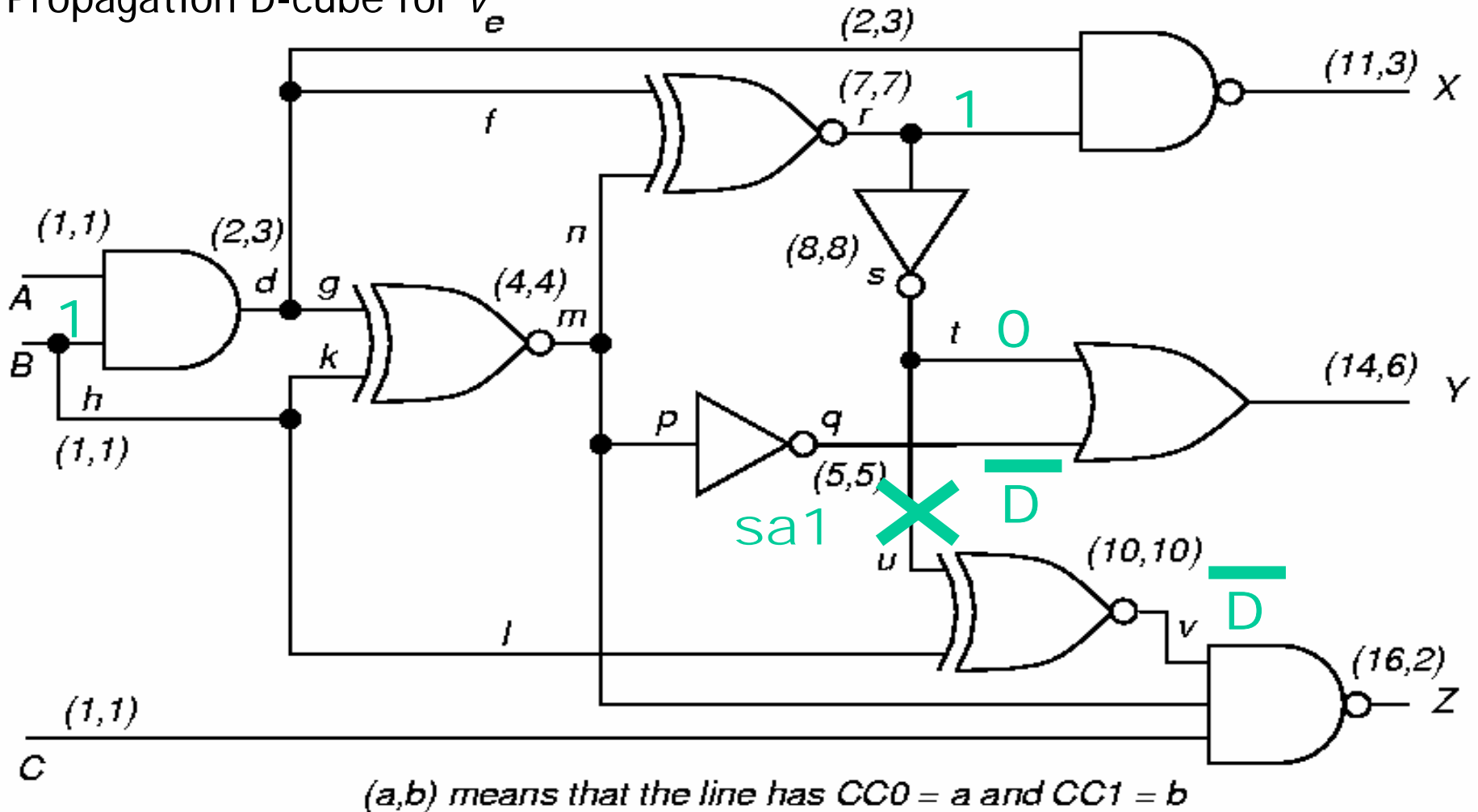
- Need alternate propagation D-cube for v





Example 7.3 - Step 3 *u sa1*

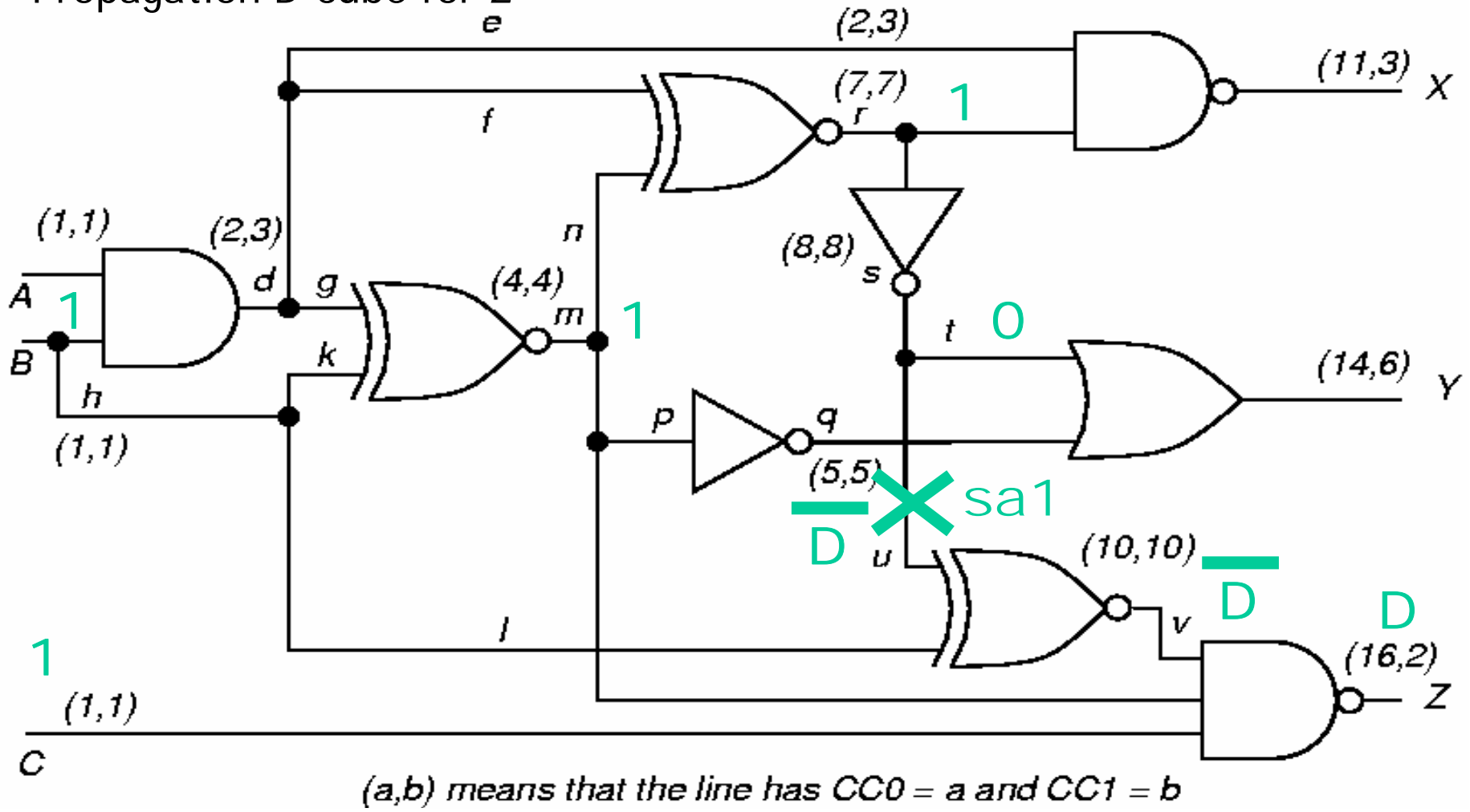
- Propagation D-cube for v_e





Example 7.3 - Step 4 *u sa1*

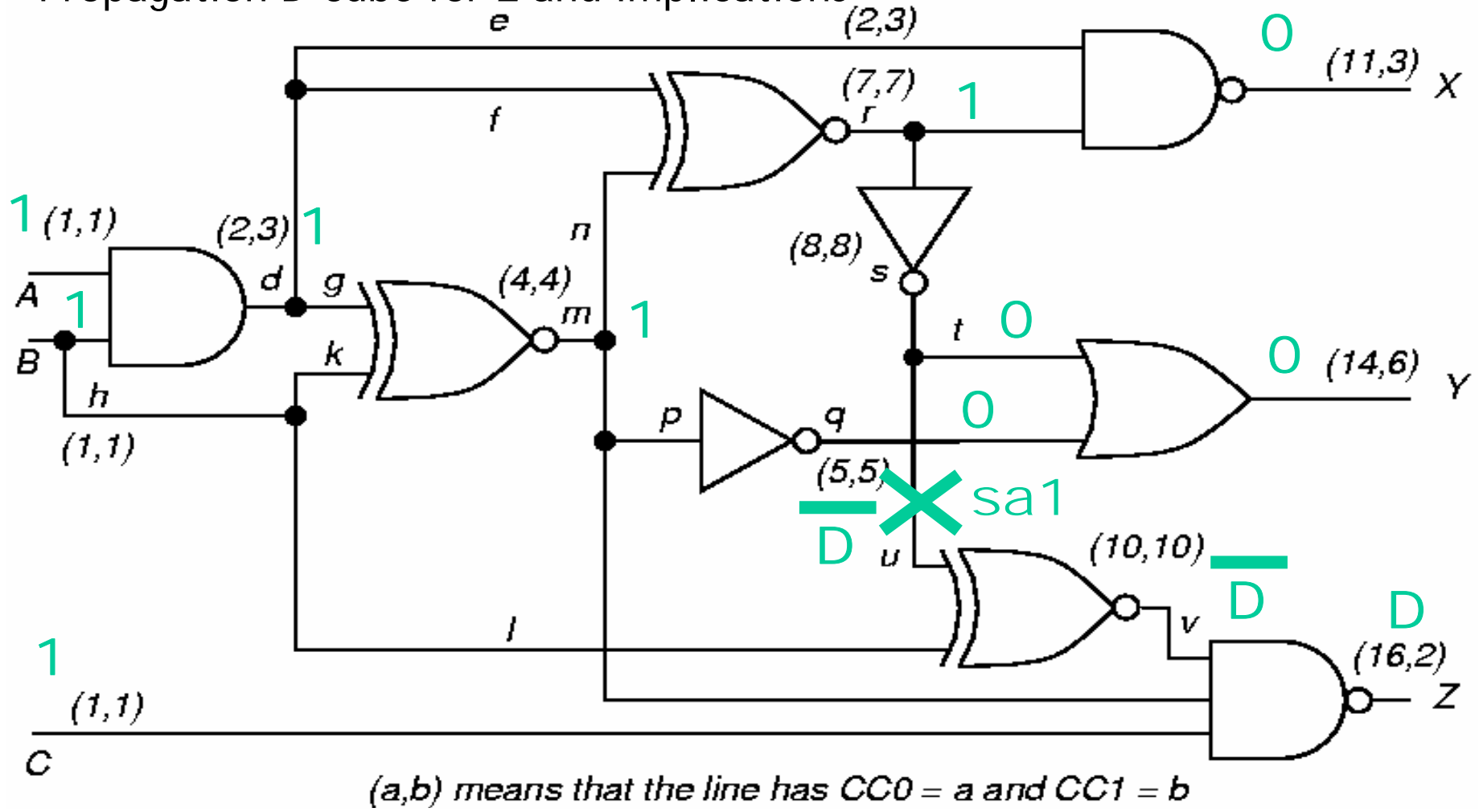
- Propagation D-cube for Z





Example 7.3 - Step 4 *u sa1*

- Propagation D-cube for Z and implications





PODEM -- Goel (1981)

- 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 (\overline{D}) to PO
 - *Backtracing*



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



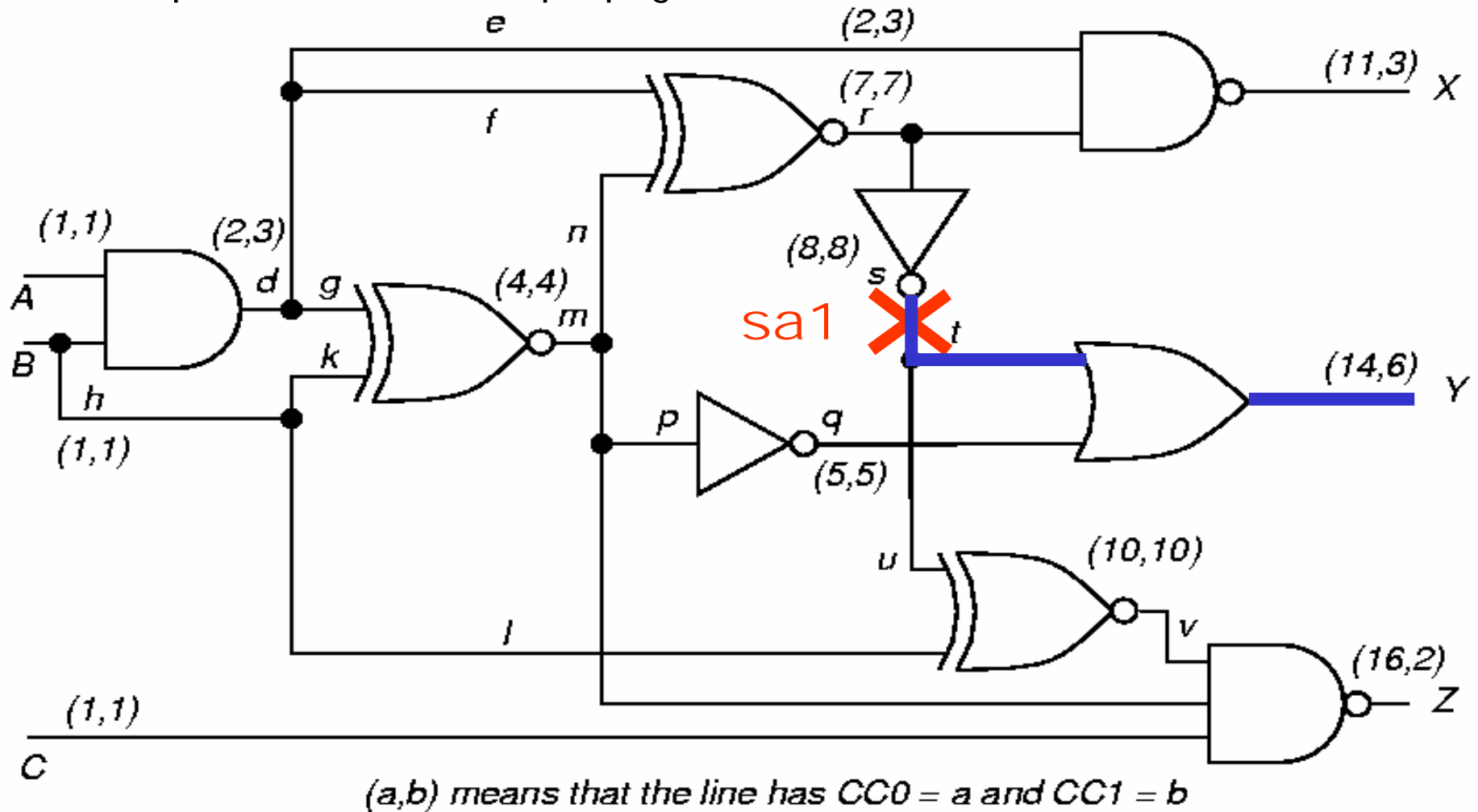
PODEM High-Level Flow

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



Example 7.3 Again

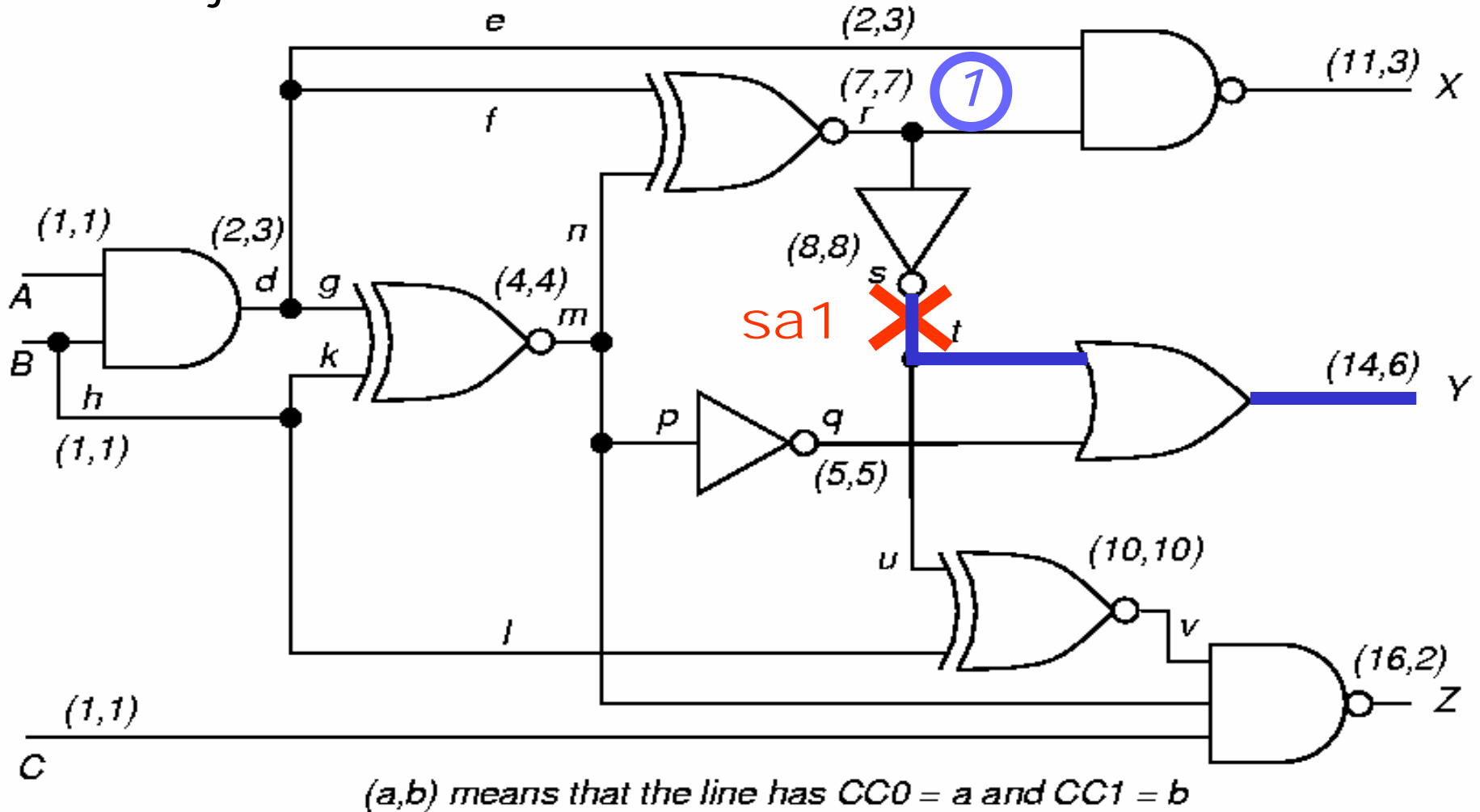
- Select path $s - Y$ for fault propagation





Example 7.3 -- Step 2 s sa1

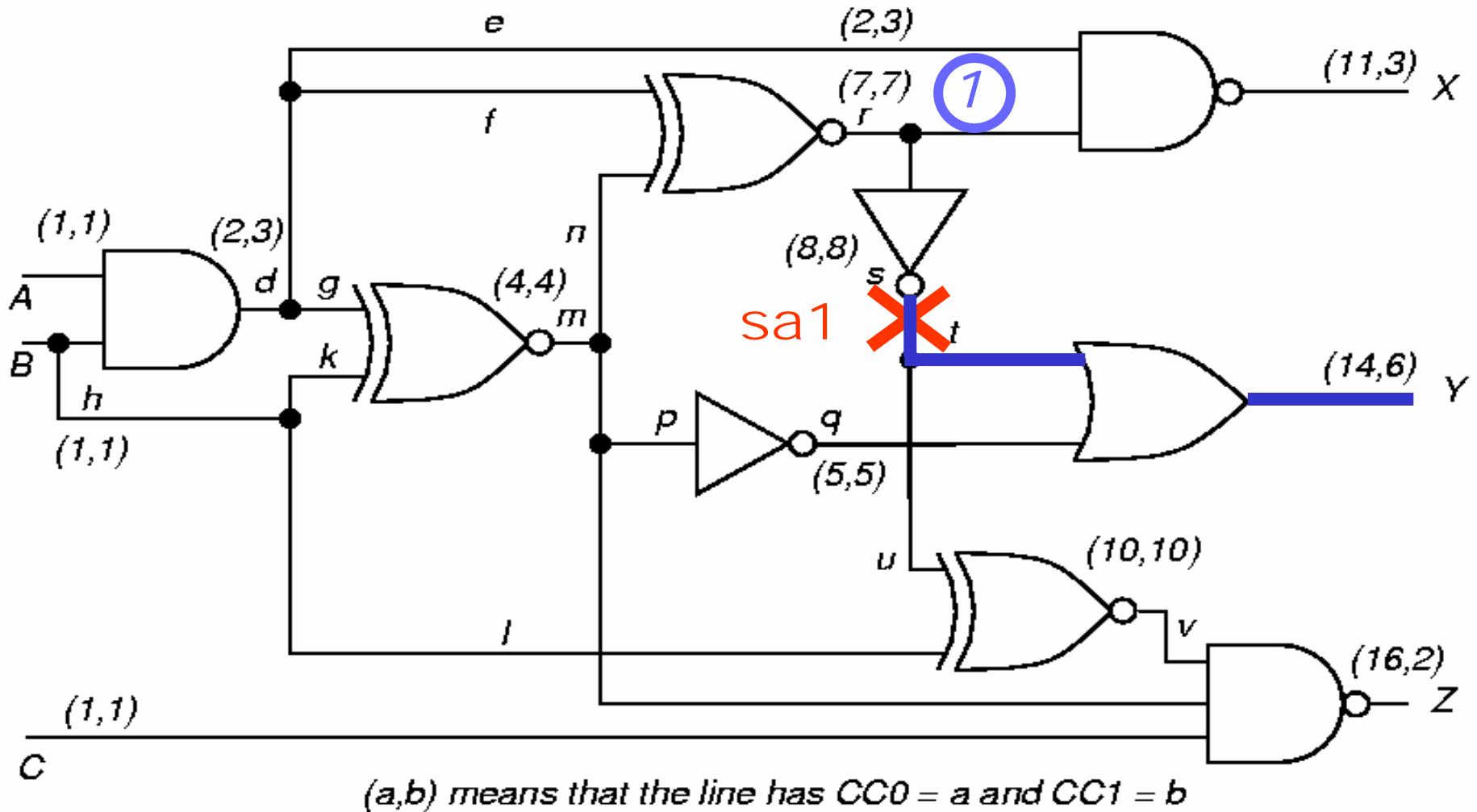
- Initial objective: Set r to 1 to excite fault





Example 7.3 -- Step 3 s sa1

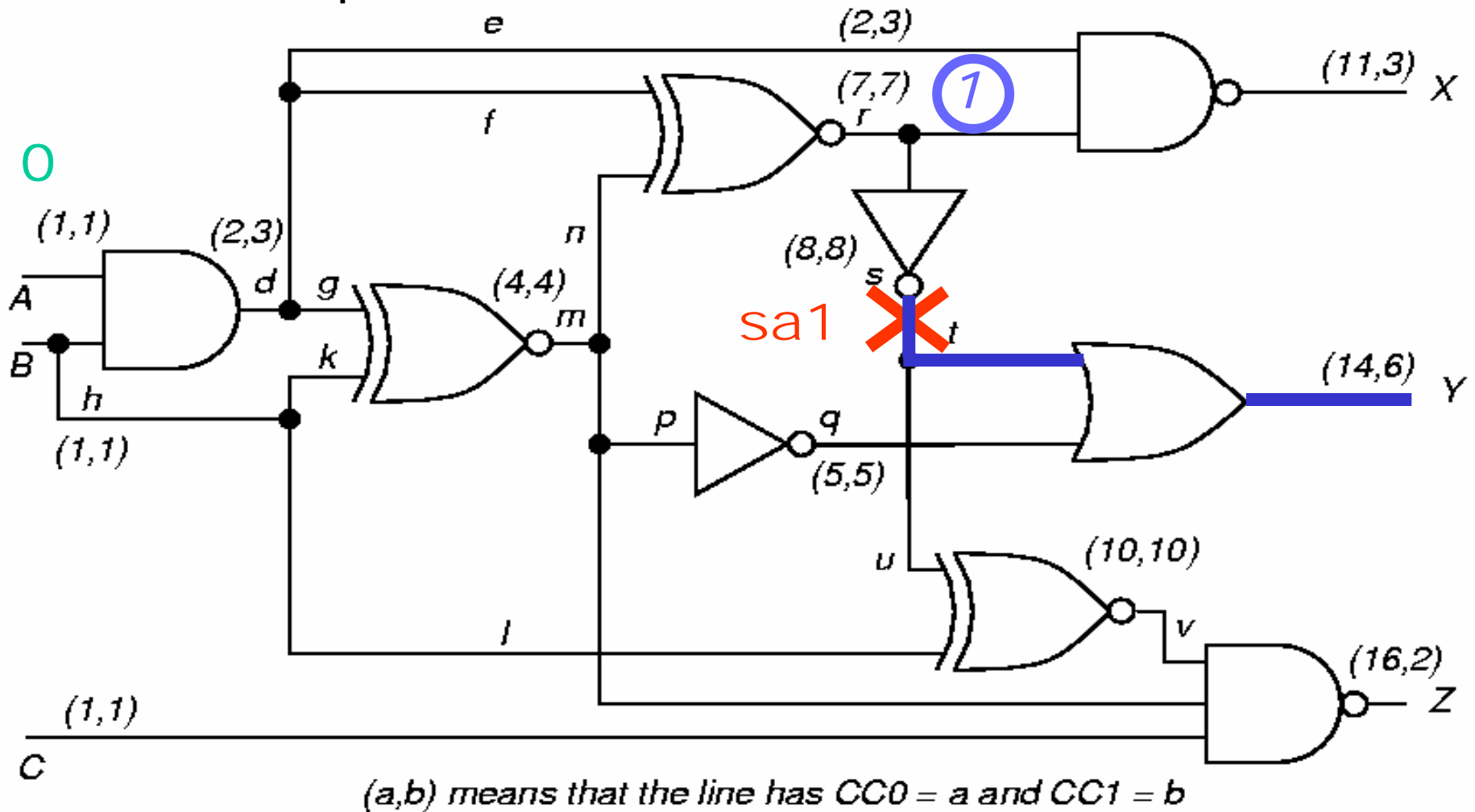
- Backtrace from r





Example 7.3 -- Step 4 s sa1

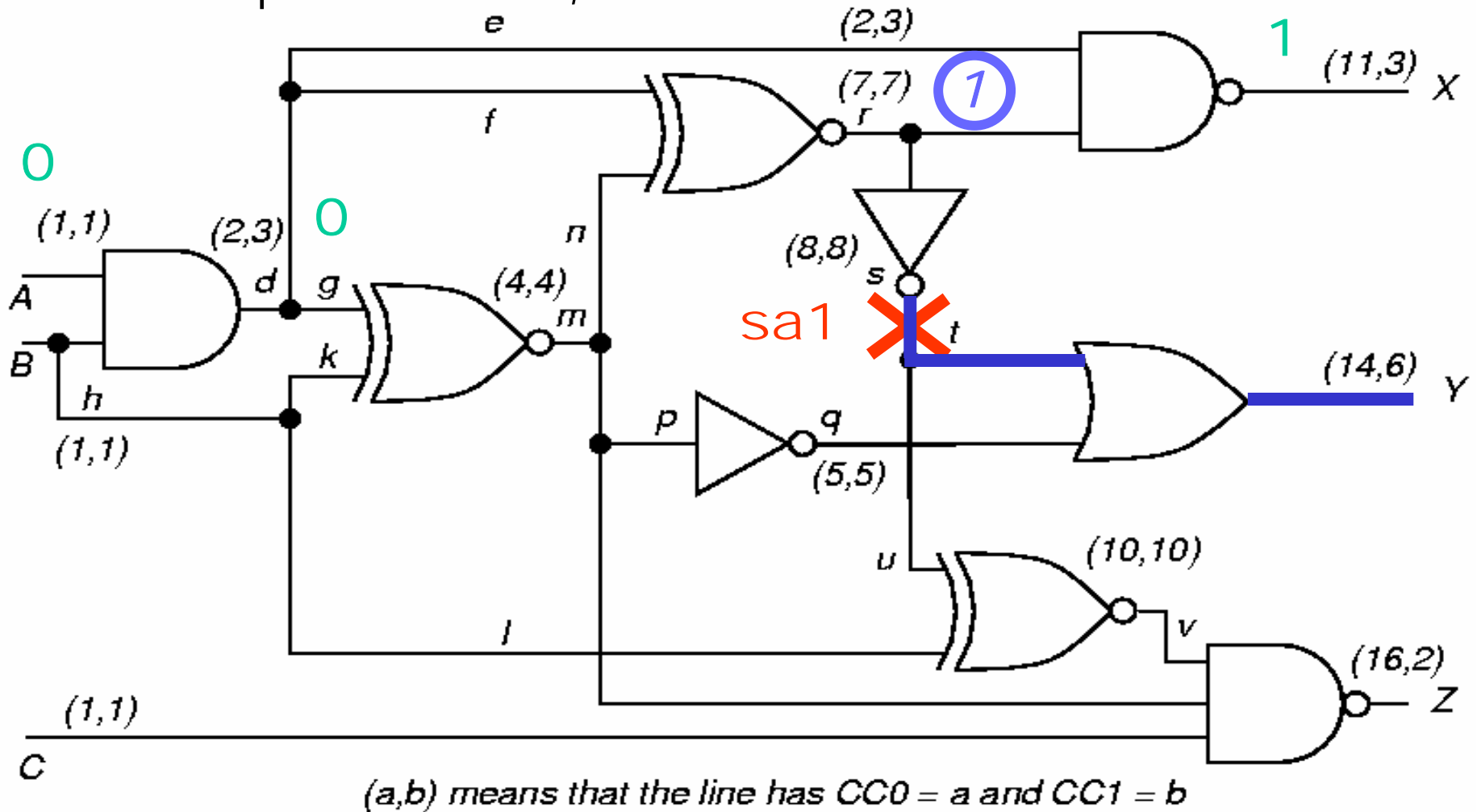
- Set $A = 0$ in implication stack





Example 7.3 -- Step 5 s sa1

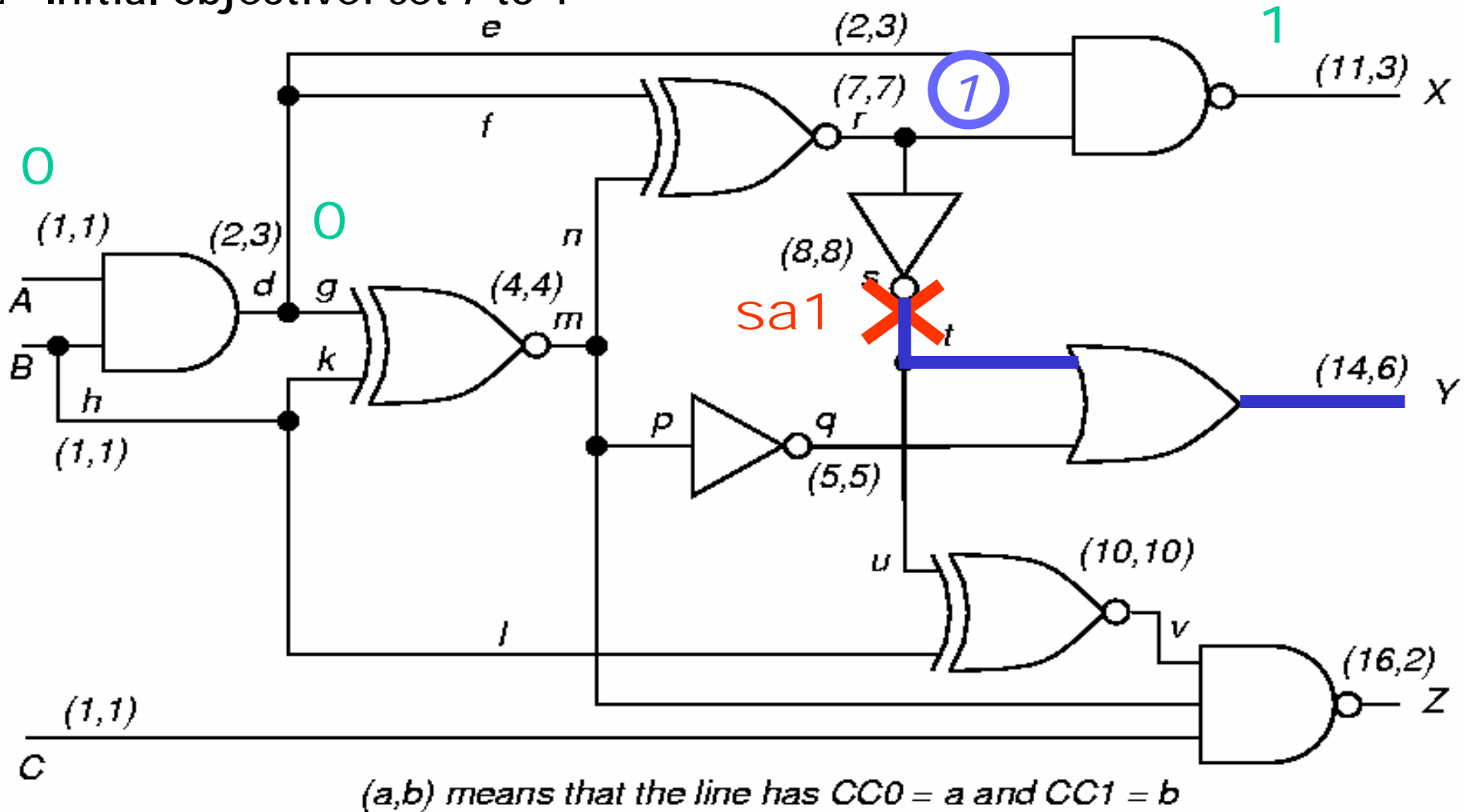
- Forward implications: $d = 0, X = 1$





Example 7.3 -- Step 6 s sa1

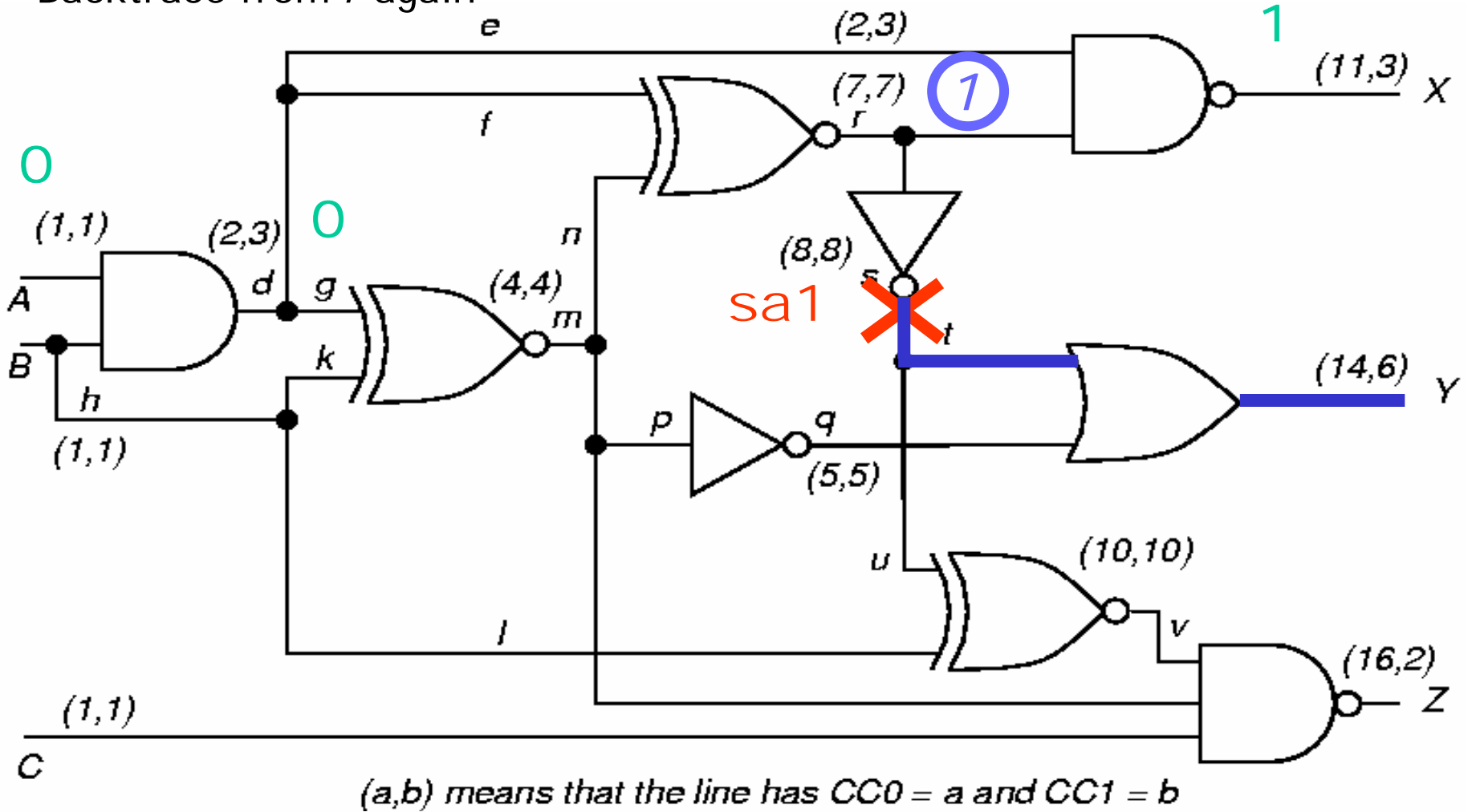
- Initial objective: set r to 1





Example 7.3 -- Step 7 s sa1

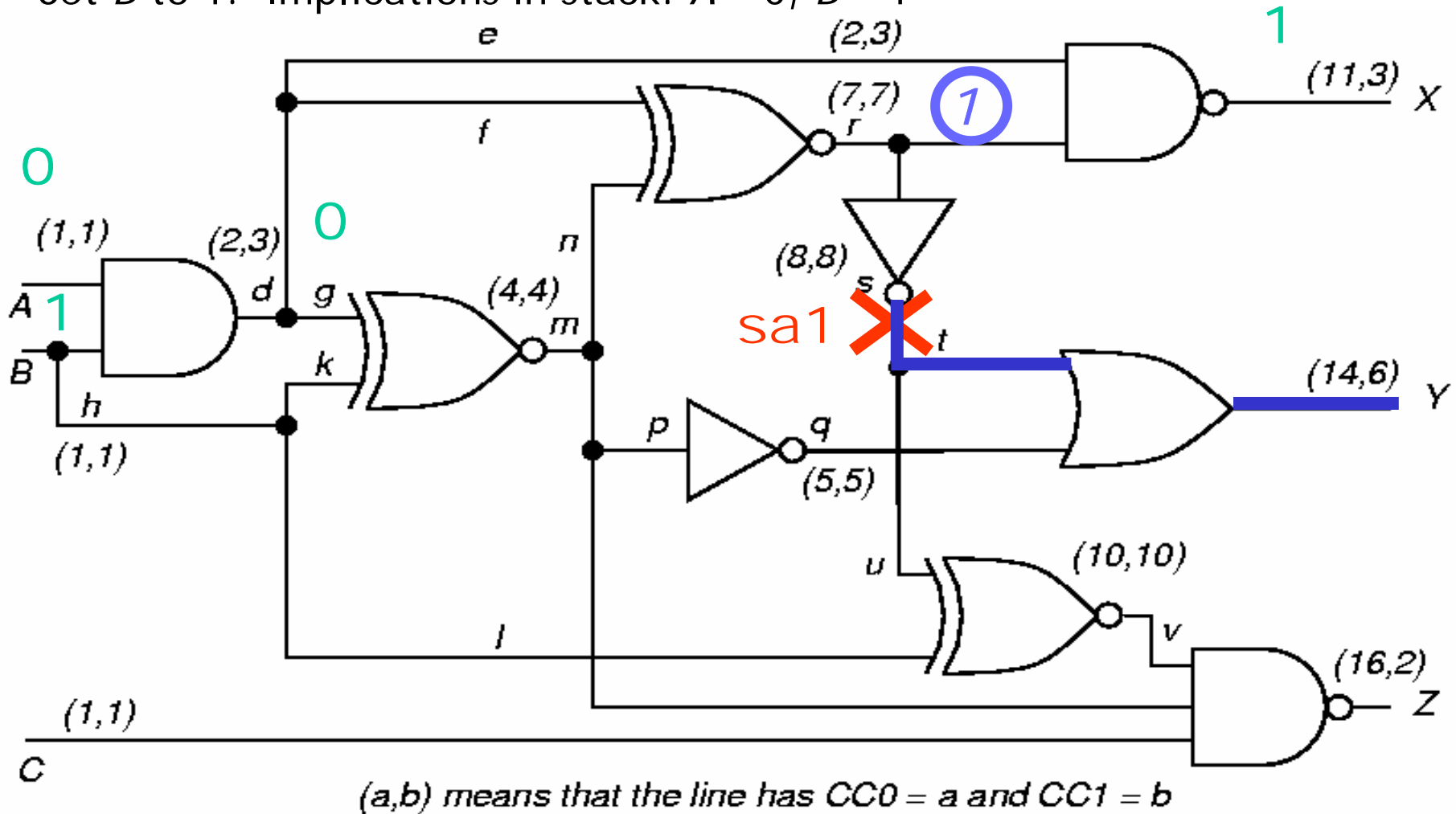
- Backtrace from r again





Example 7.3 -- Step 8 s sa1

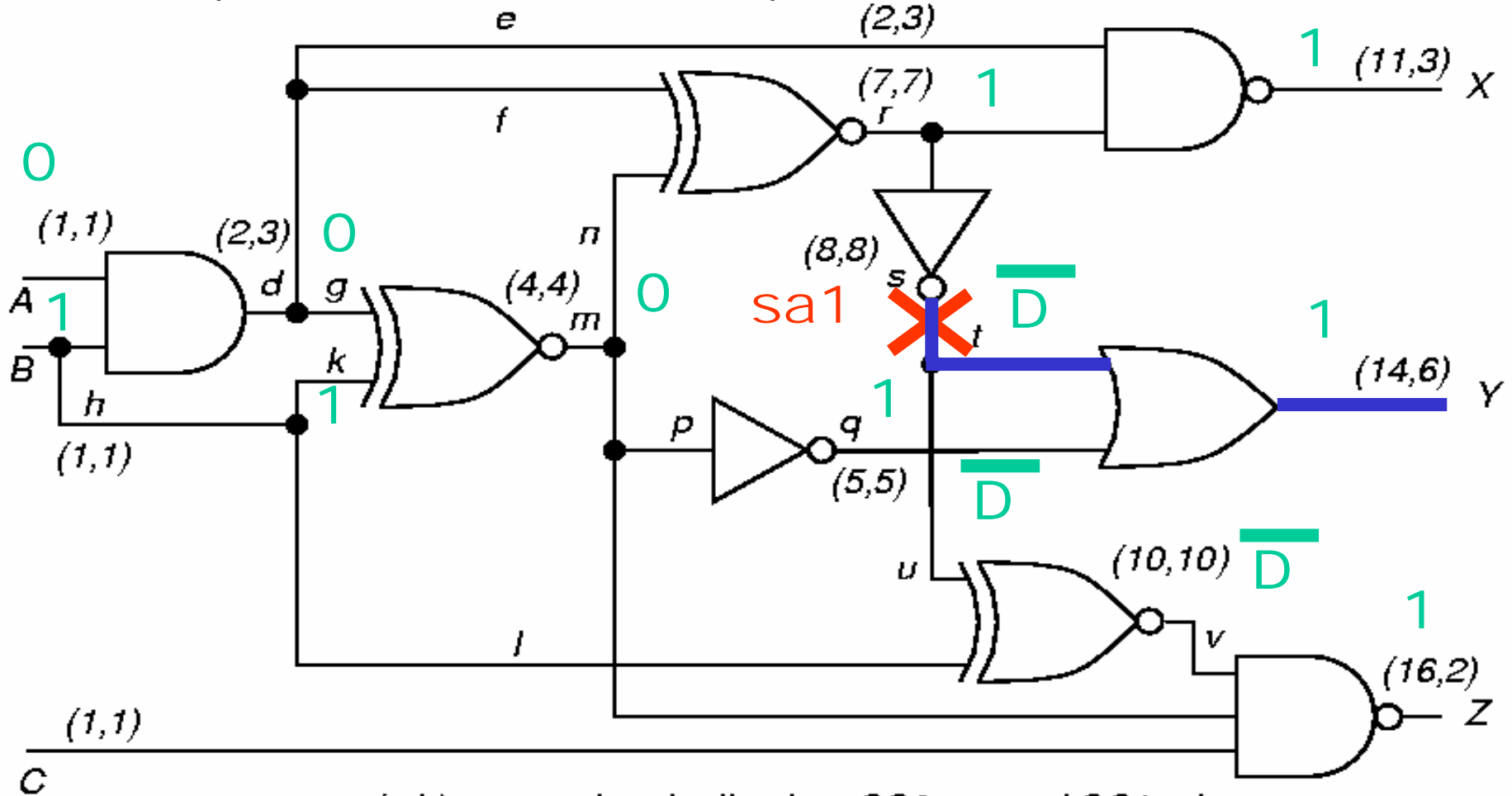
- Set B to 1. Implications in stack: $A = 0, B = 1$





Example 7.3 -- Step 9 s sa1

- Forward implications: $k = 1, m = 0, r = 1, q = 1, Y = 1, s = \overline{D}, u = \overline{D}, v = \overline{D}, Z = 1$

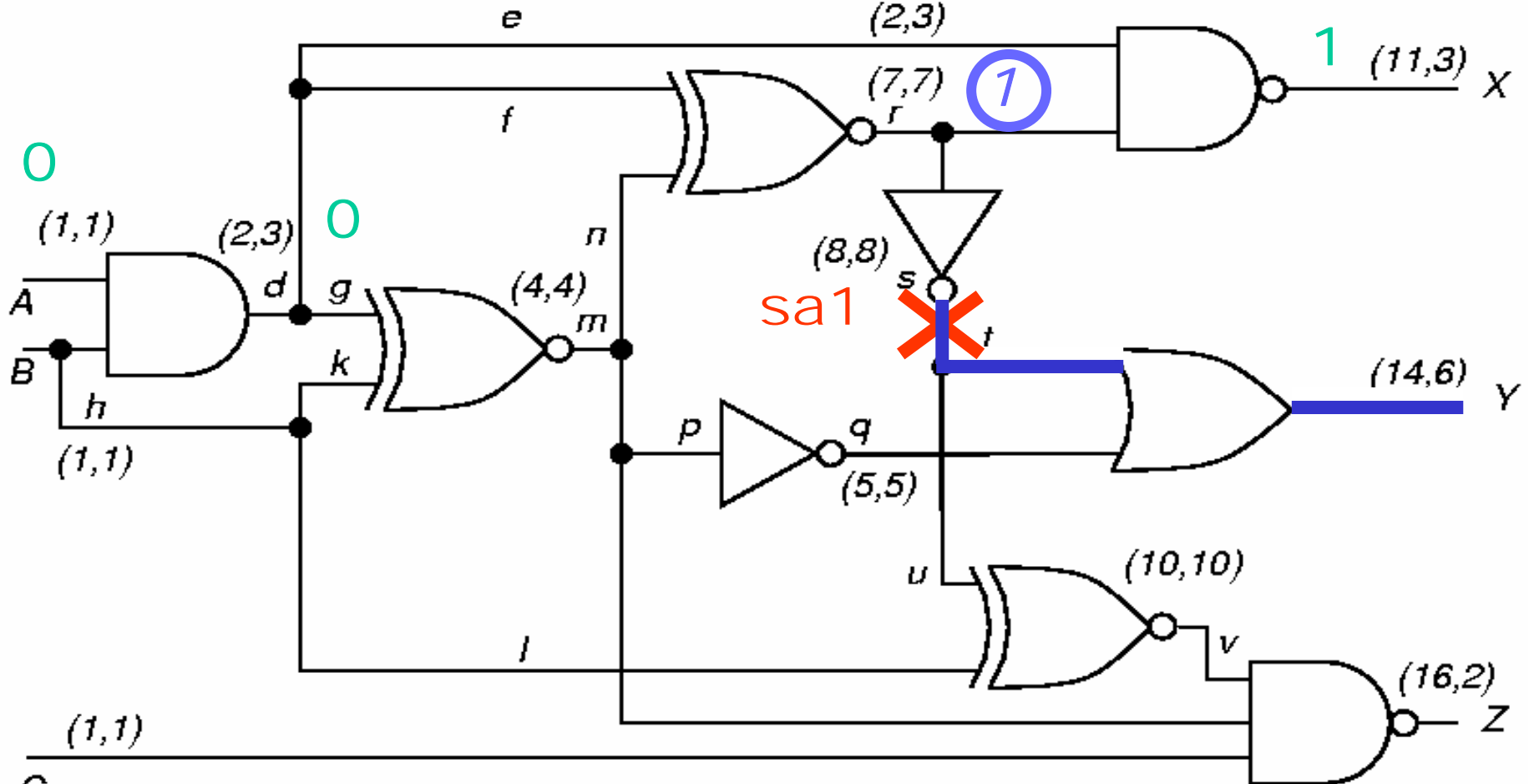


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Backtrack -- Step 10 s sa1

- *X-PATH-CHECK* shows paths $s - Y$ and $s - u - v - Z$ blocked (*D-frontier* disappeared)

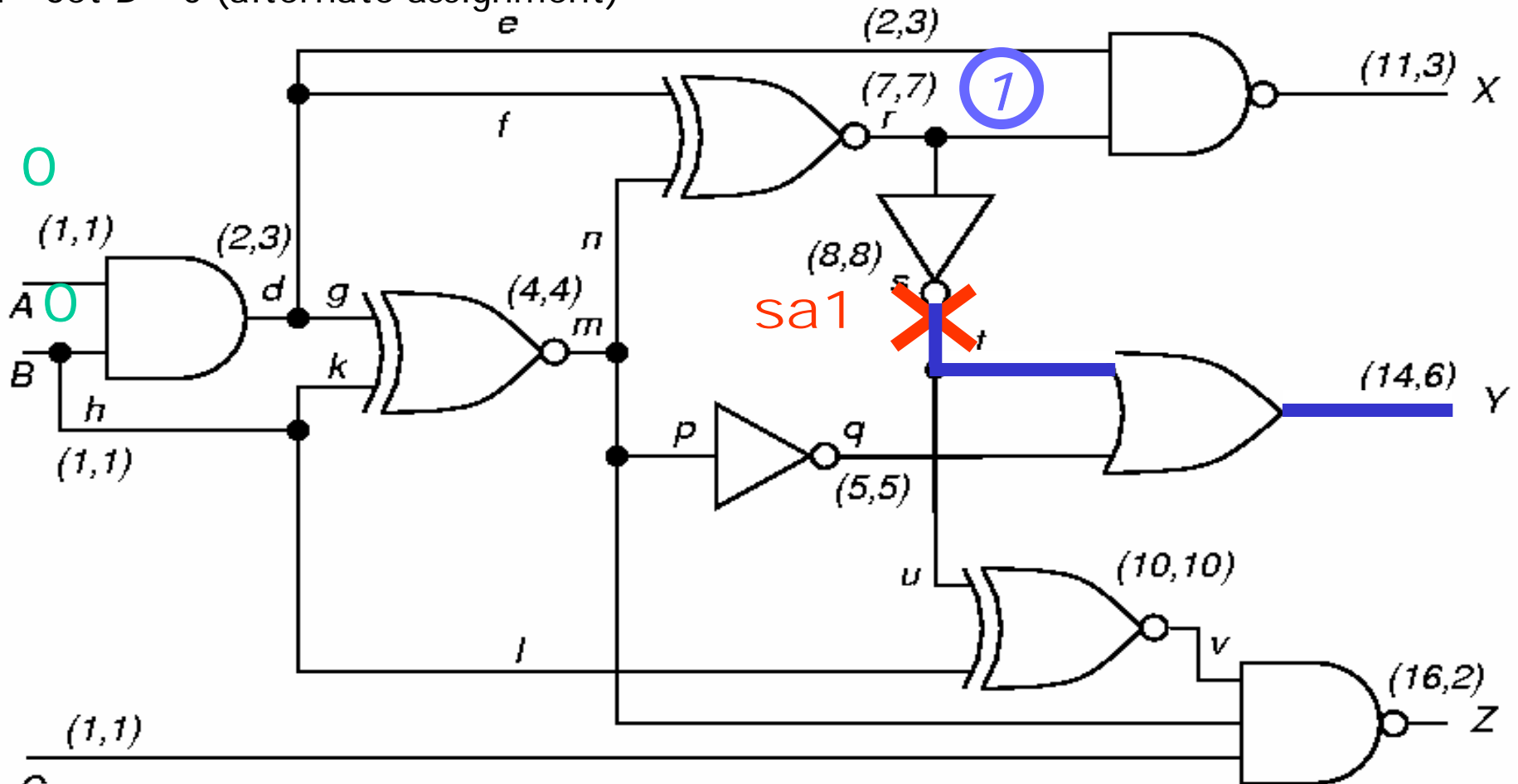


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Step 11 -- s sa1

- Set $B = 0$ (alternate assignment)



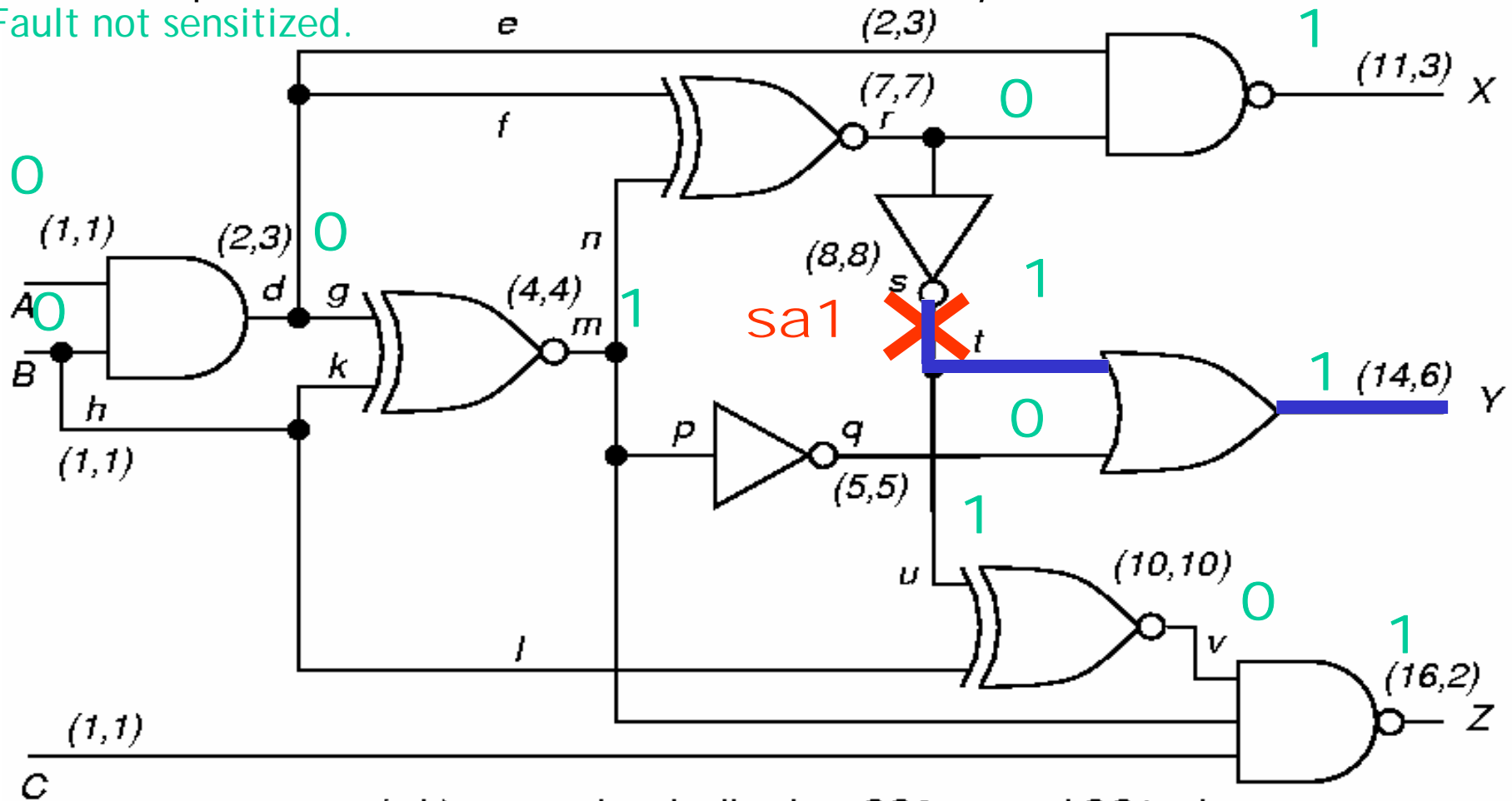
(a,b) means that the line has CC0 = a and CC1 = b



Backtrack -- s sa1

a Forward implications: $d = 0, X = 1, m = 1, r = 0, s = 1, q = 0, Y = 1, v = 0, Z = 1$.

Fault not sensitized.

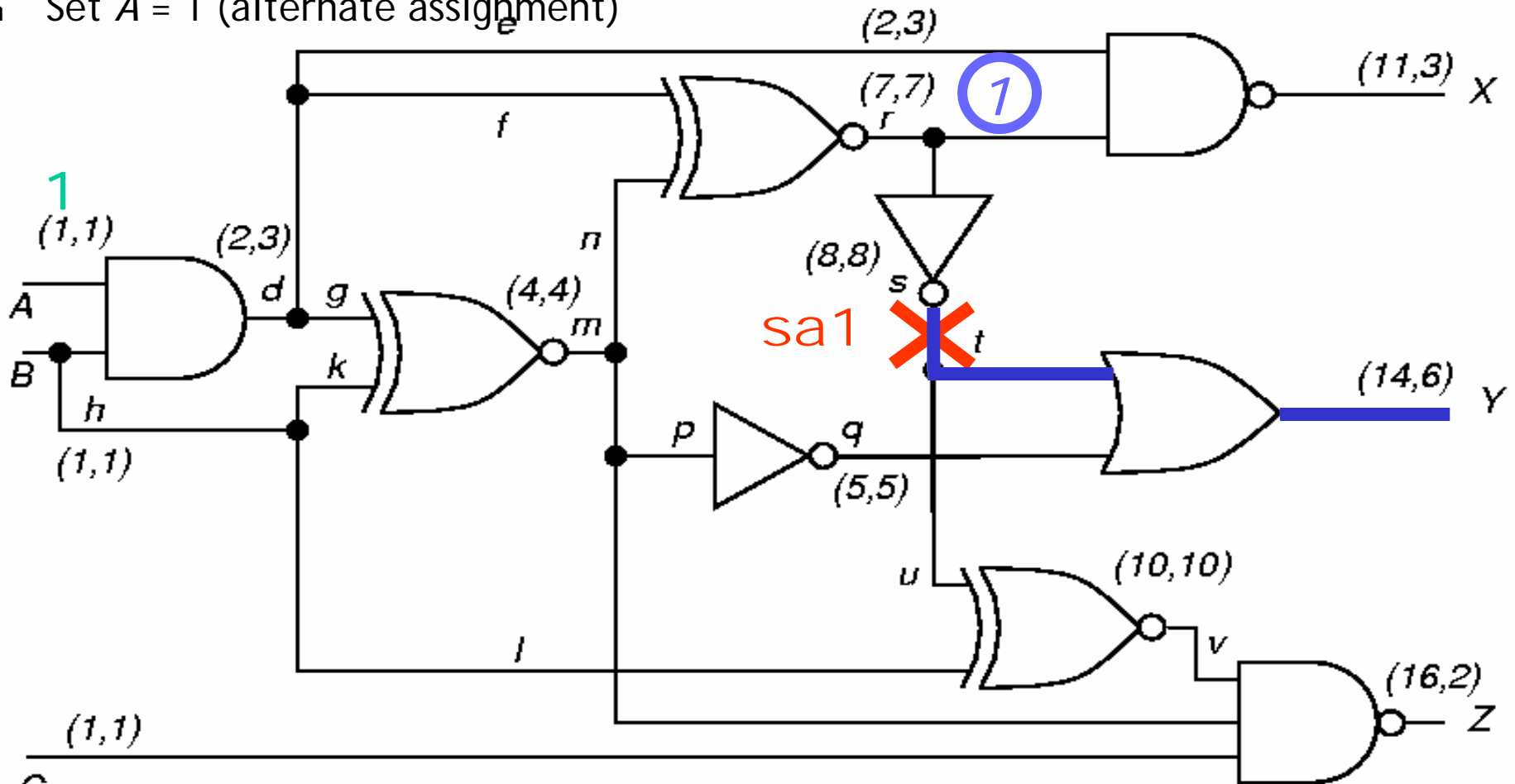


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Step 13 -- s sa1

- Set $A = 1$ (alternate assignment)

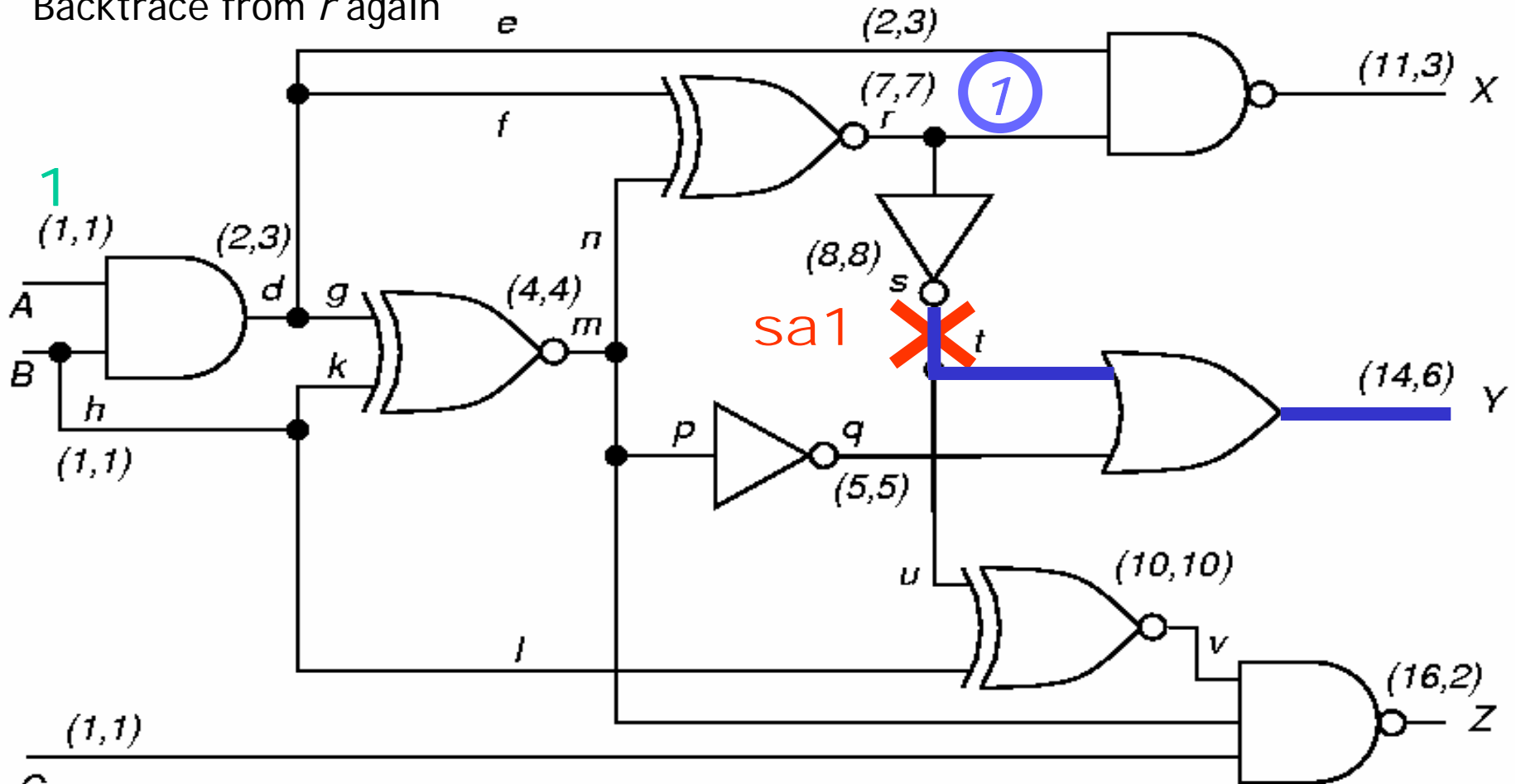


(a,b) means that the line has CC0 = a and CC1 = b



Step 14 -- s sa1

- Backtrace from r again

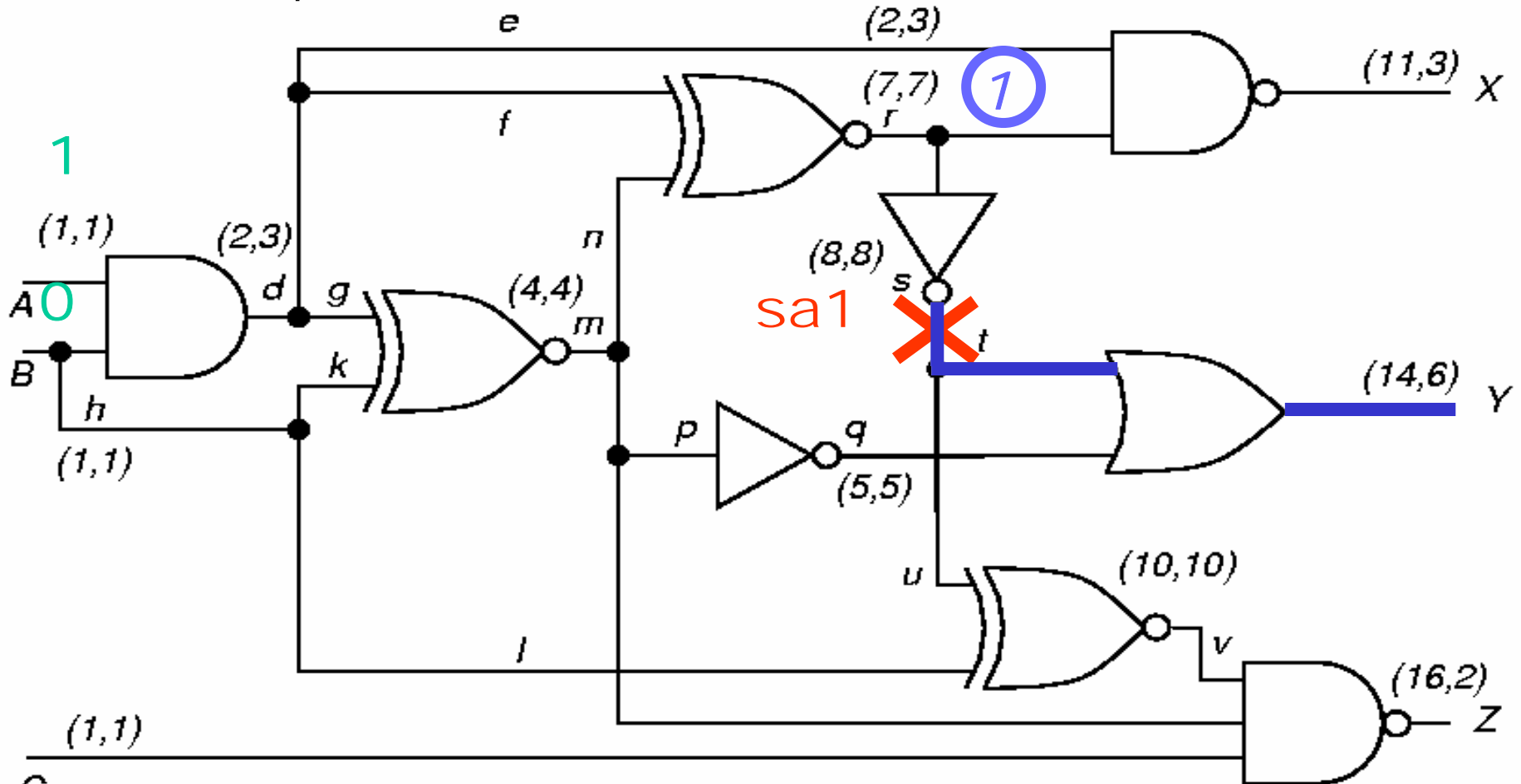


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Step 15 -- s sa1

- Set $B = 0$. Implications in stack: $A = 1, B = 0$

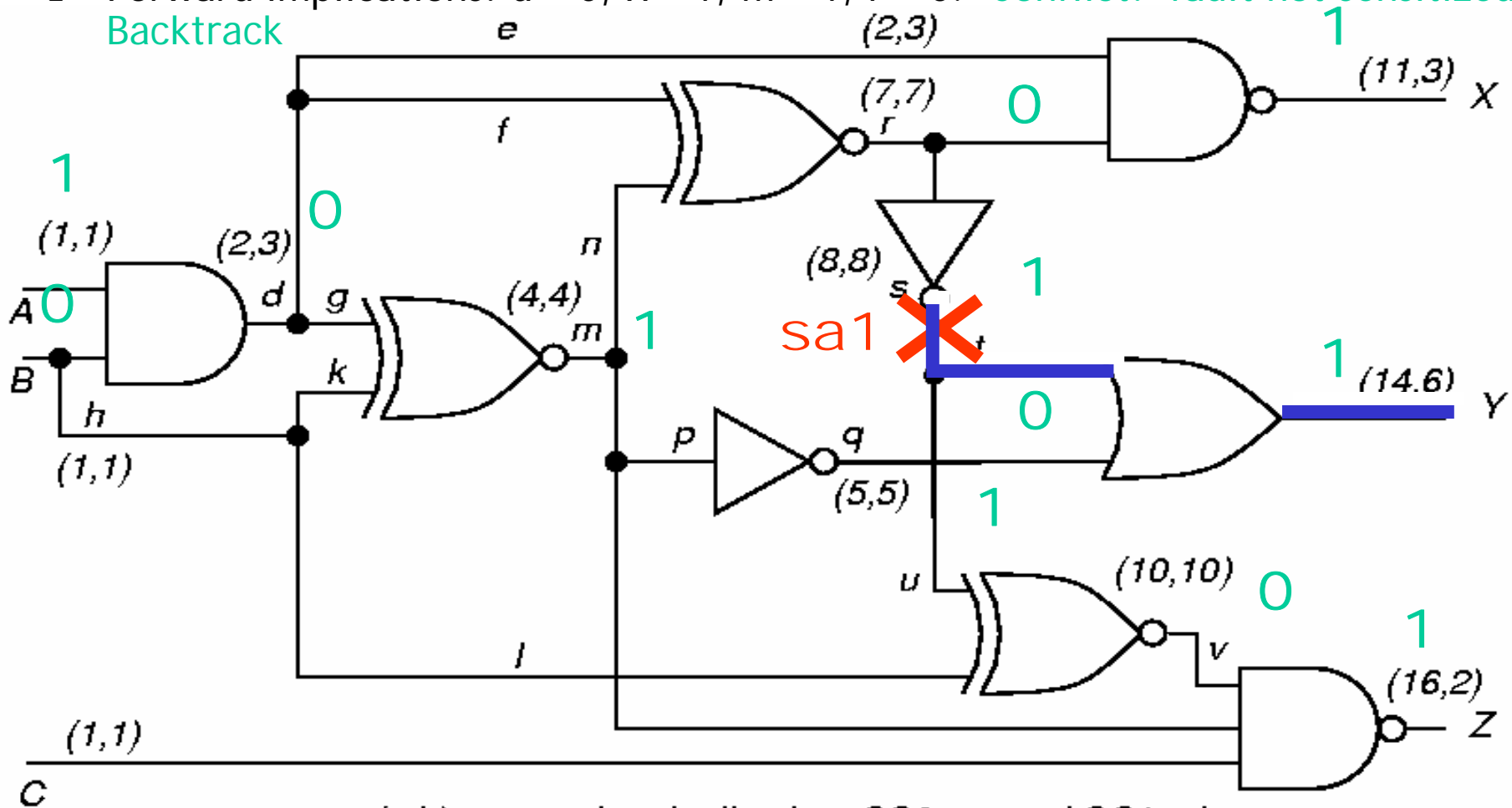


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Backtrack -- s sa1

- Forward implications: $d = 0, X = 1, m = 1, r = 0$. Conflict: fault not sensitized. Backtrack

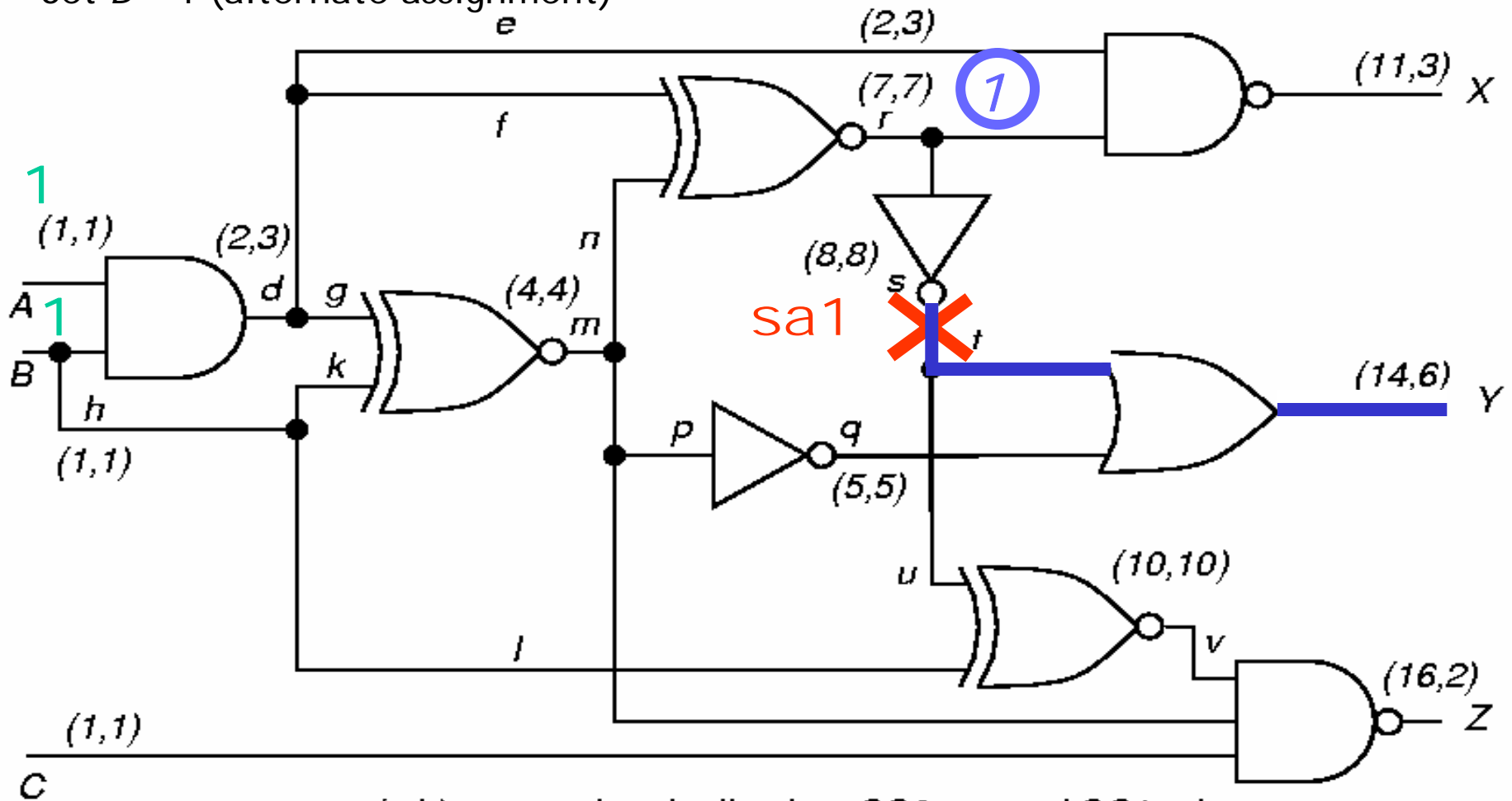


(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Step 17 -- s sa1

- Set $B = 1$ (alternate assignment)

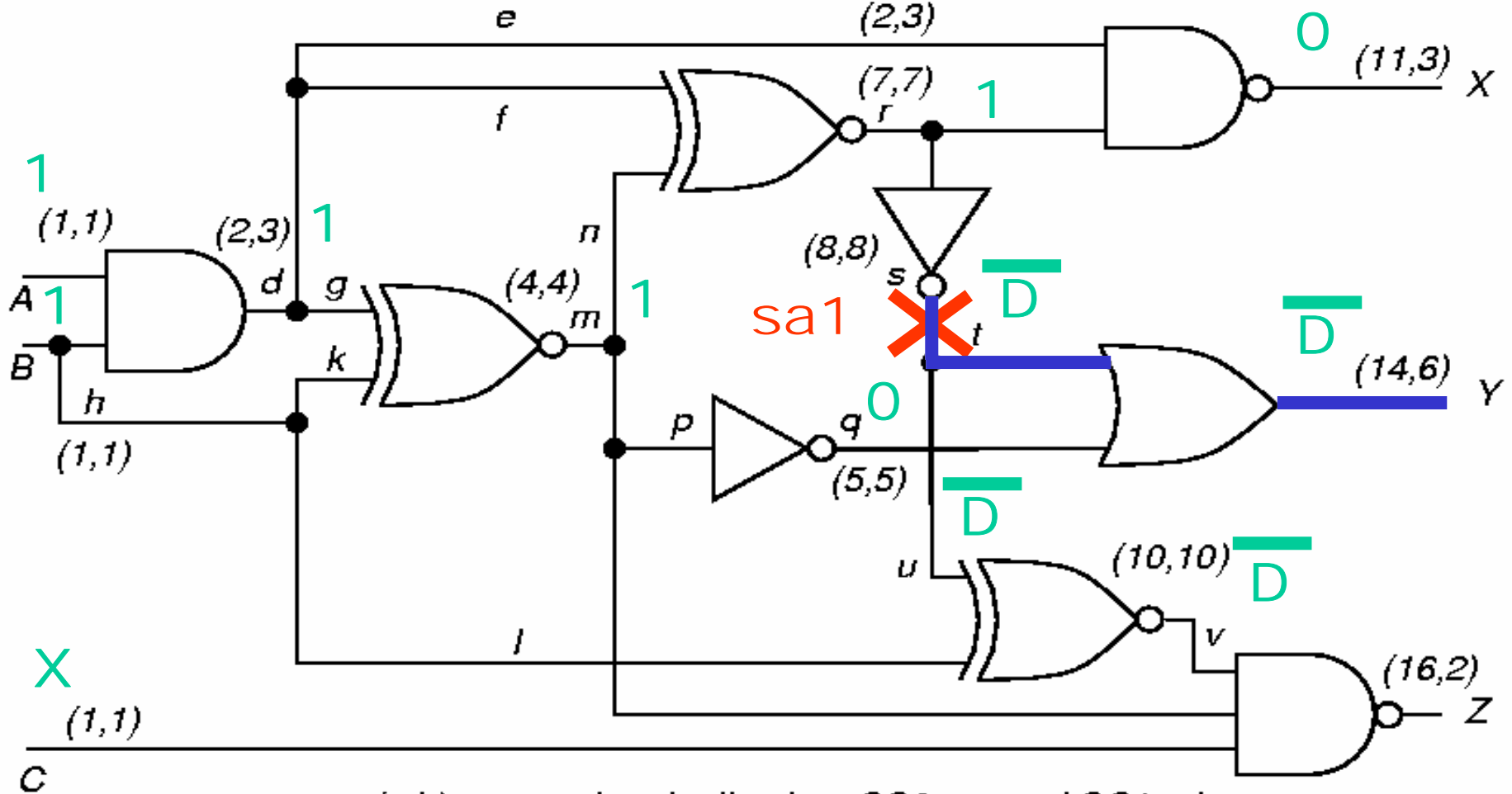


(a,b) means that the line has CC0 = a and CC1 = b



Fault Tested -- Step 18 s sa1

- Forward implications: $d = 1, m = 1, r = 1, q = 0, s = \overline{D}, v = \overline{D}, X = 0, Y = \overline{D}$



(a,b) means that the line has $CC0 = a$ and $CC1 = b$



Backtrace (s, v_s) Pseudo-Code

```
 $V = V_s$ ;  
while ( $s$  is a gate output)  
    if ( $s$  is NAND or INVERTER or NOR)  $v = \overline{V}$ ;  
    if (objective requires setting all inputs)  
        select unassigned input  $a$  of  $s$  with hardest controllability to value  $v$ ;  
    else  
        select unassigned input  $a$  of  $s$  with easiest controllability to value  $v$ ;  
     $s = a$ ;  
return ( $s, v$ ) /* Gate and value to be assigned */;
```



Objective Selection Code

```
if (gate  $g$  is unassigned) return ( $g, \overline{v}$ );  
select a gate  $P$  from the D-frontier;  
select an unassigned input  $I$  of  $P$ ;  
if (gate  $g$  has controlling value)  
     $c =$  controlling input value of  $g$ ;  
else if (0 value easier to get at input of XOR/EQUIV gate)  
     $c = 1$ ;  
else  $c = 0$ ;  
return ( $I, \overline{c}$ );
```



PODEM Algorithm

```
while (no fault effect at POs)
  if (xpathcheck (D-frontier)
      (l, vl) = Objective (fault, vfault);
      (pi, vpi) = Backtrace (l, vl);
      Imply (pi, vpi);
      if (PODEM (fault, vfault) == SUCCESS) return (SUCCESS);
      (pi, vpi) = Backtrack ();
      Imply (pi, vpi);
      if (PODEM (fault, vfault) == SUCCESS) return (SUCCESS);
      Imply (pi, "X");
      return (FAILURE);
  else if (implication stack exhausted)
    return (FAILURE);
  else Backtrack ();
return (SUCCESS);
```



Summary

- D-ALG - First complete ATPG algorithm
 - *D-Cube*
 - *D-Calculus*
 - *Implications* - forward and backward
 - *Implication stack*
 - *Backup*
- PODEM
 - Expand decision tree only around PIs
 - Use *X-PATH-CHECK* to see if *D-frontier* exists
 - *Objectives* -- bring ATPG closer to getting \overline{D} to PO
 - *Backtracing*



Metodologie di progetto HW

Il test di circuiti digitali

Sequential ATPG



Overview

- Motivation
- Sequential circuit ATPG
- An example test generation
- Time-frame expansion
 - Nine-valued logic
 - ATPG implementation and drivability
 - Complexity of ATPG
 - Cycle-free and cyclic circuits
- Test generations systems
 - Classification
 - Forward time test generator - FASTEST
 - General comments
 - Simulation based test generators - contest, strategate
- Summary

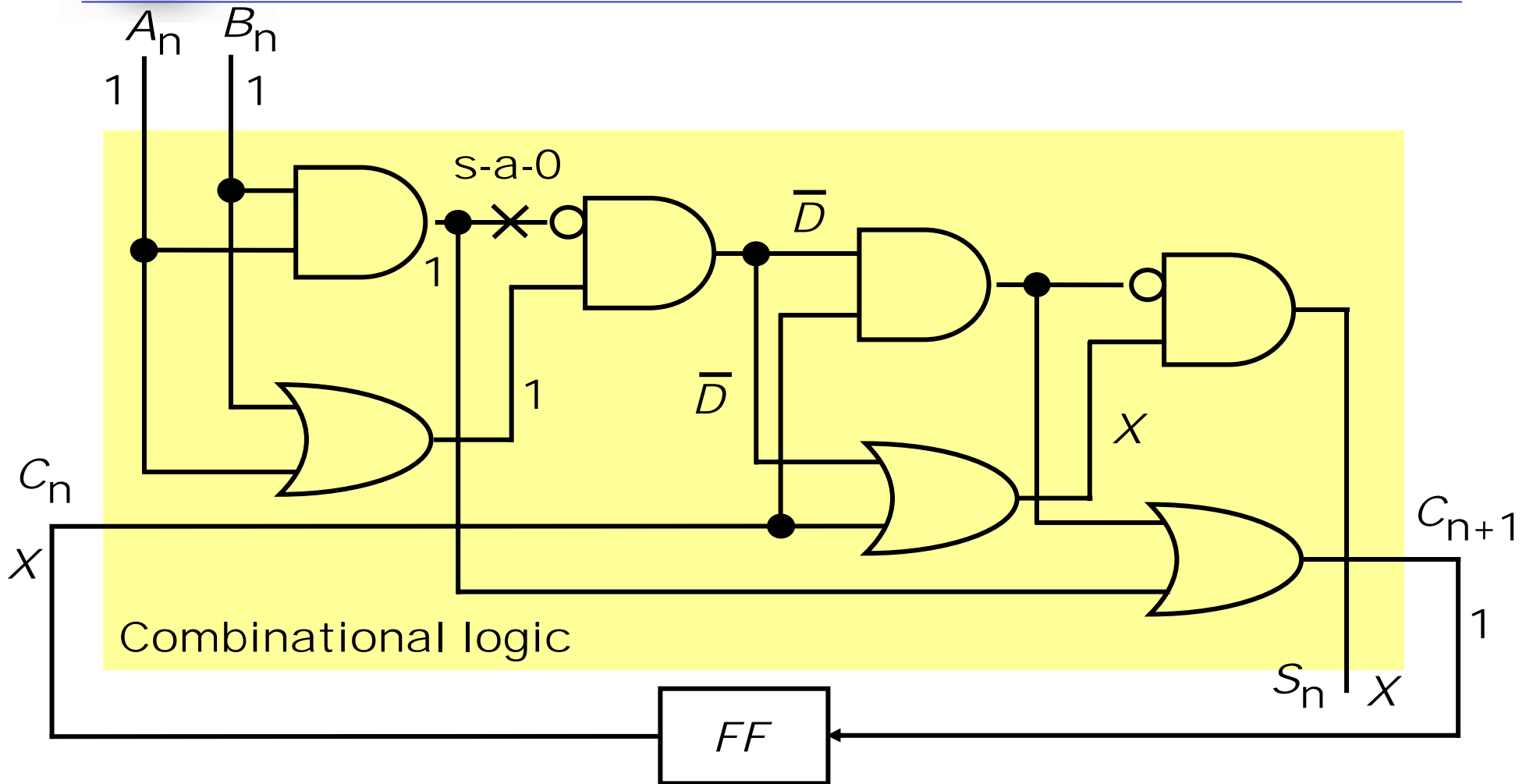


Sequential Circuits

- ❑ A sequential circuit has memory in addition to combinational logic.
- ❑ Test for a fault in a sequential circuit is a sequence of vectors, which
 - Initializes the circuit to a known state
 - Activates the fault, and
 - Propagates the fault effect to a primary output
- ❑ Methods of sequential circuit ATPG
 - Time-frame expansion methods
 - Simulation-based methods

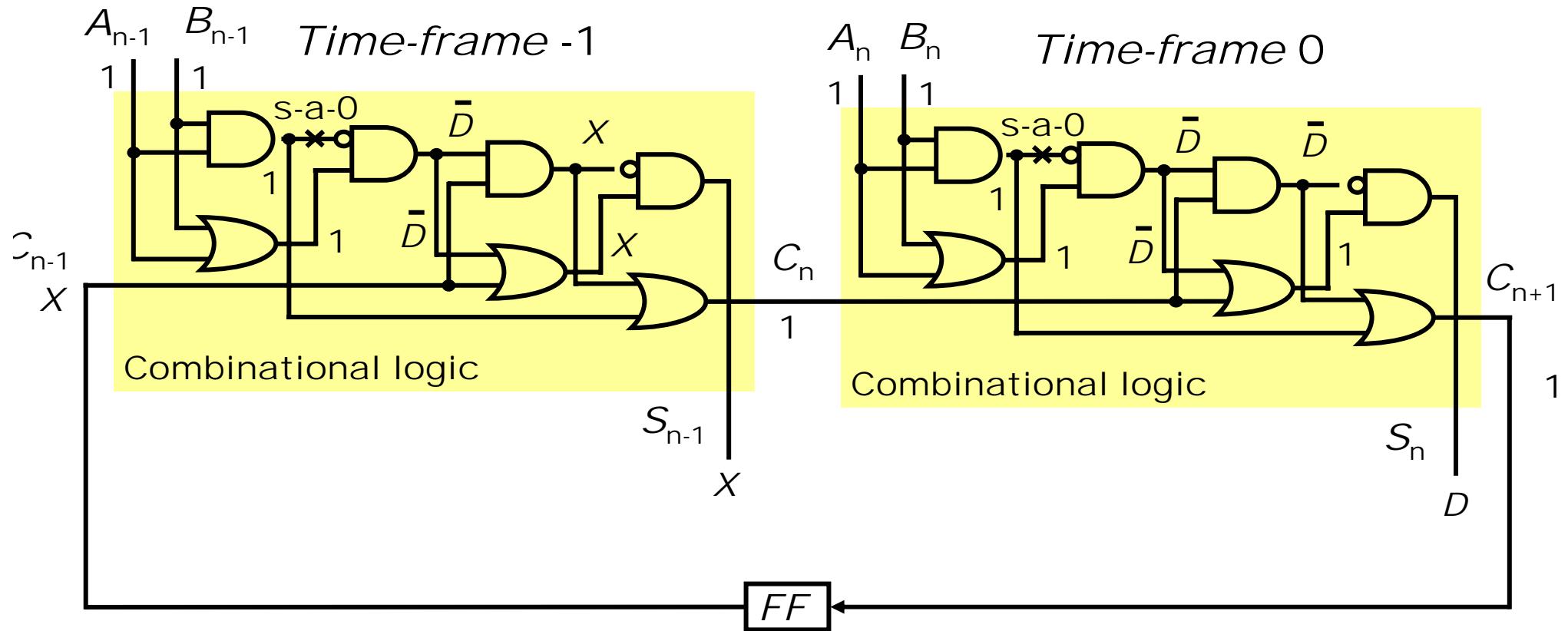


Example: A Serial Adder





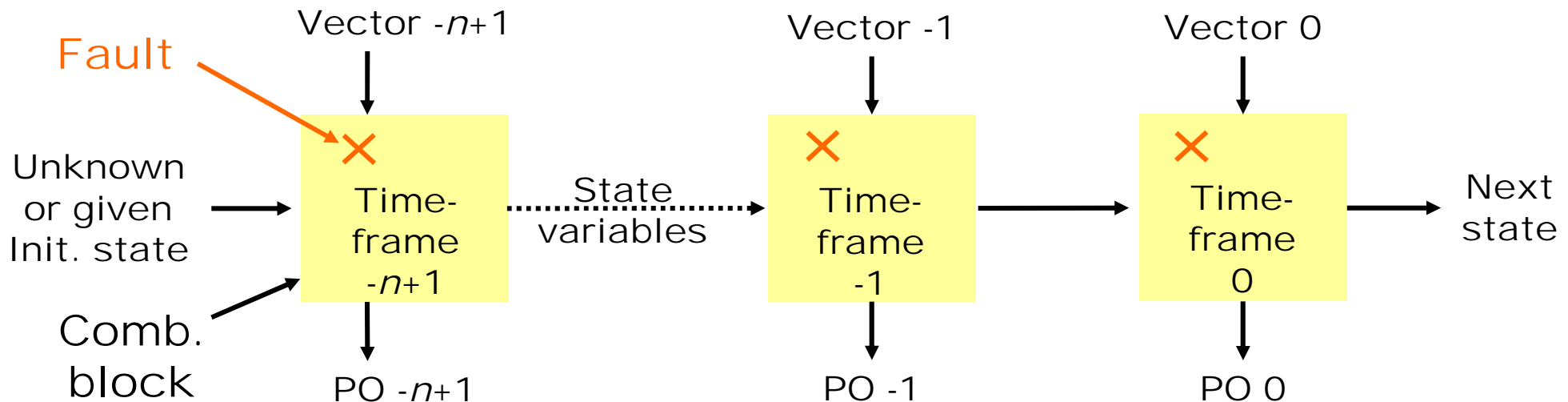
Time-Frame Expansion





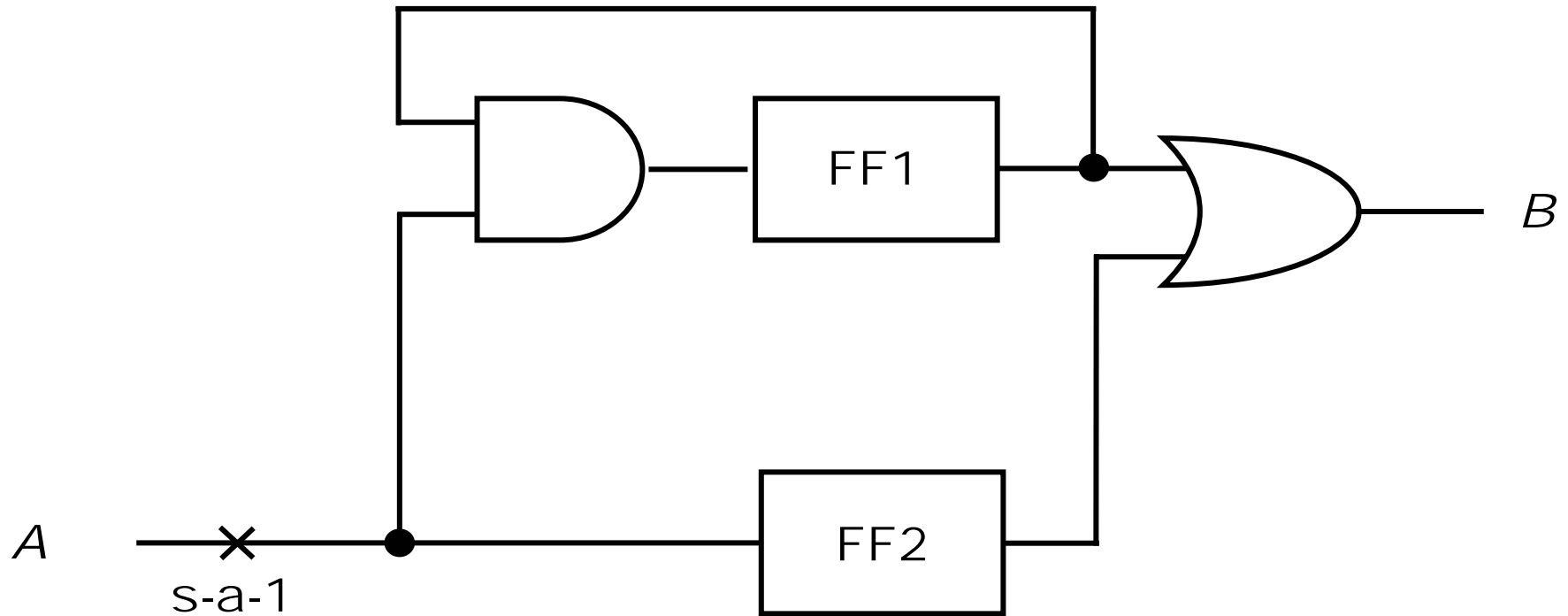
Concept of Time-Frames

- If the test sequence for a single stuck-at fault contains n vectors,
 - Replicate combinational logic block n times
 - Place fault in each block
 - Generate a test for the multiple stuck-at fault using combinational ATPG with 9-valued logic



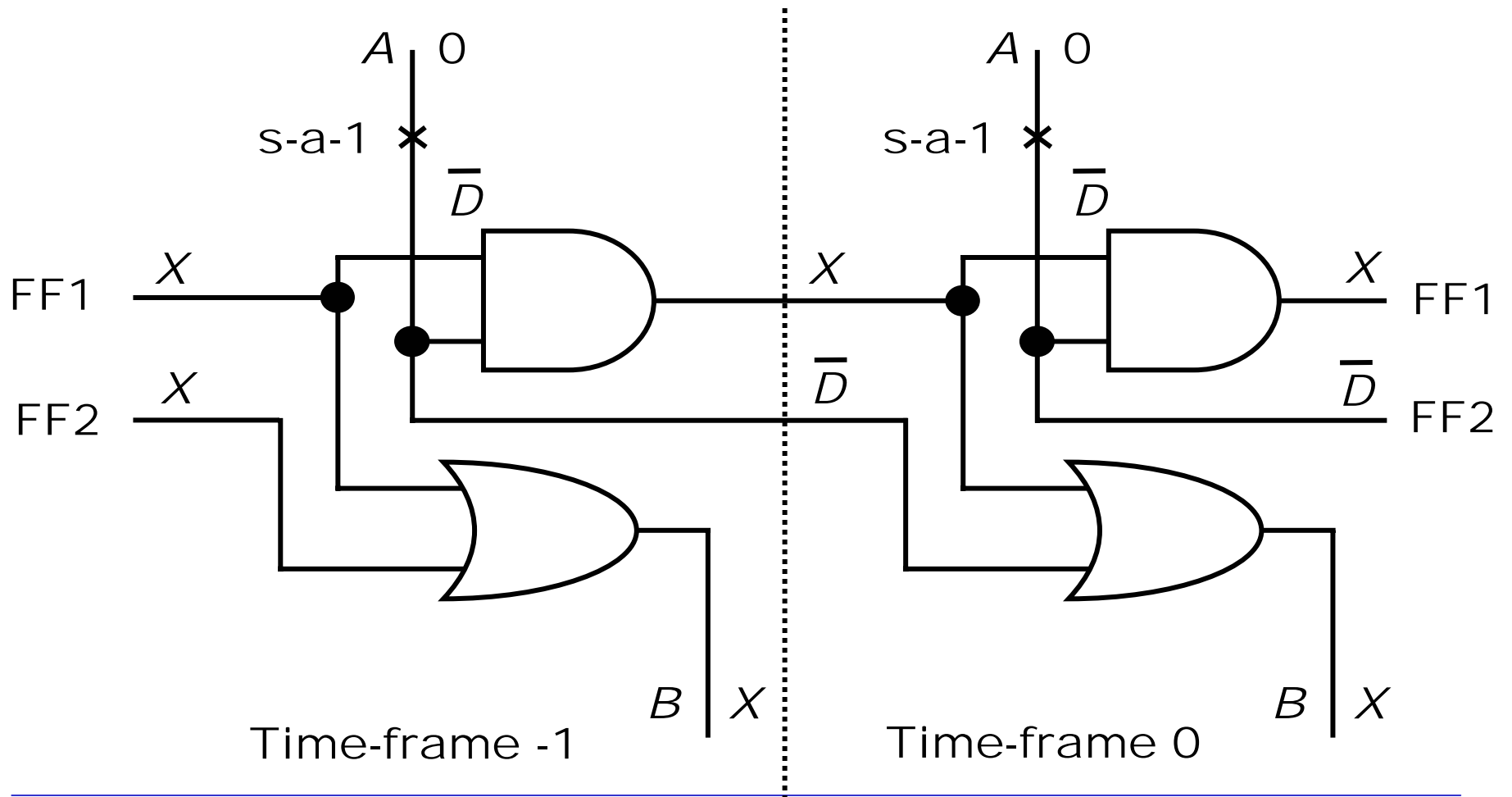


Example for Logic Systems





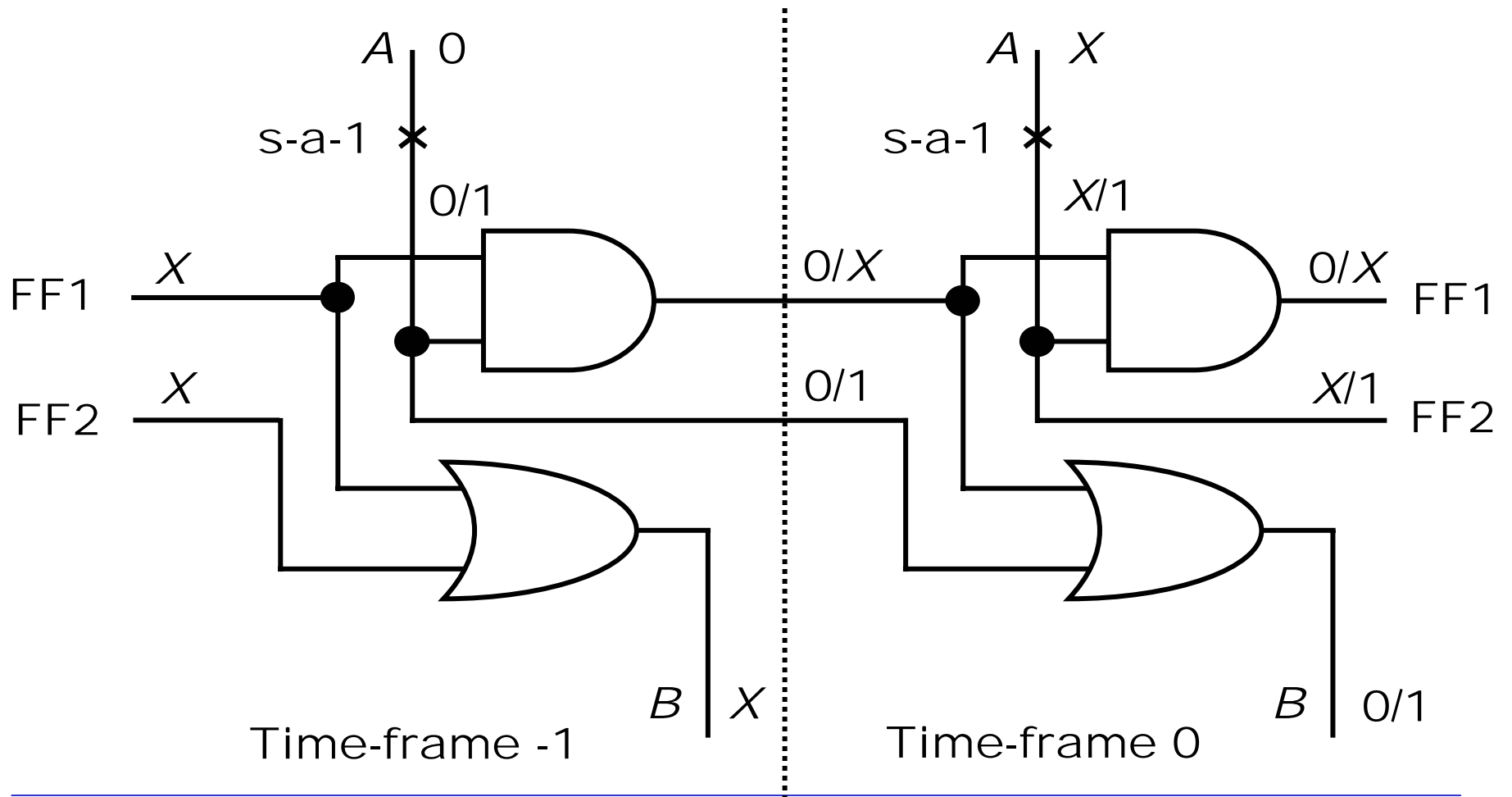
Five-Valued Logic (Roth) $0, 1, D, \bar{D}, X$





Nine-Valued Logic (Muth)

0, 1, 1/0, 0/1, 1/X, 0/X, X/0, X/1, X





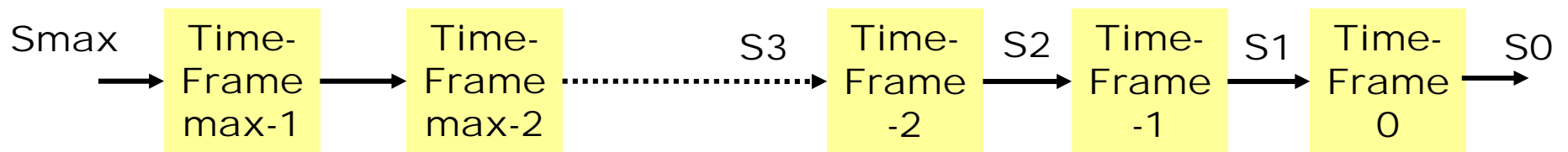
An implementation of ATPG

- ❑ Select a PO for fault detection based on drivability analysis.
 - ❑ Place a logic value, 1/0 or 0/1, depending on fault type and number of inversions.
 - ❑ Justify the output value from PIs, considering all necessary paths and adding backward time-frames.
 - ❑ If justification is impossible, then use drivability to select another PO and repeat justification.
 - ❑ If the procedure fails for all reachable POs, then the fault is *untestable*.
 - ❑ If 1/0 or 0/1 cannot be justified at any PO, but 1/X or 0/X can be justified, the the fault is *potentially detectable*.
-



Complexity of ATPG

- Synchronous circuit -- All flip-flops controlled by clocks; PI and PO synchronized with clock:
 - Cycle-free circuit - No feedback among flip-flops: Test generation for a fault needs no more than $dseq + 1$ time-frames, where $dseq$ is the sequential depth.
 - Cyclic circuit - Contains feedback among flip-flops: May need 9^{Nff} time-frames, where Nff is the number of flip-flops.
- Asynchronous circuit - Higher complexity!



$max = \text{Number of distinct vectors with 9-valued elements} = 9^{Nff}$

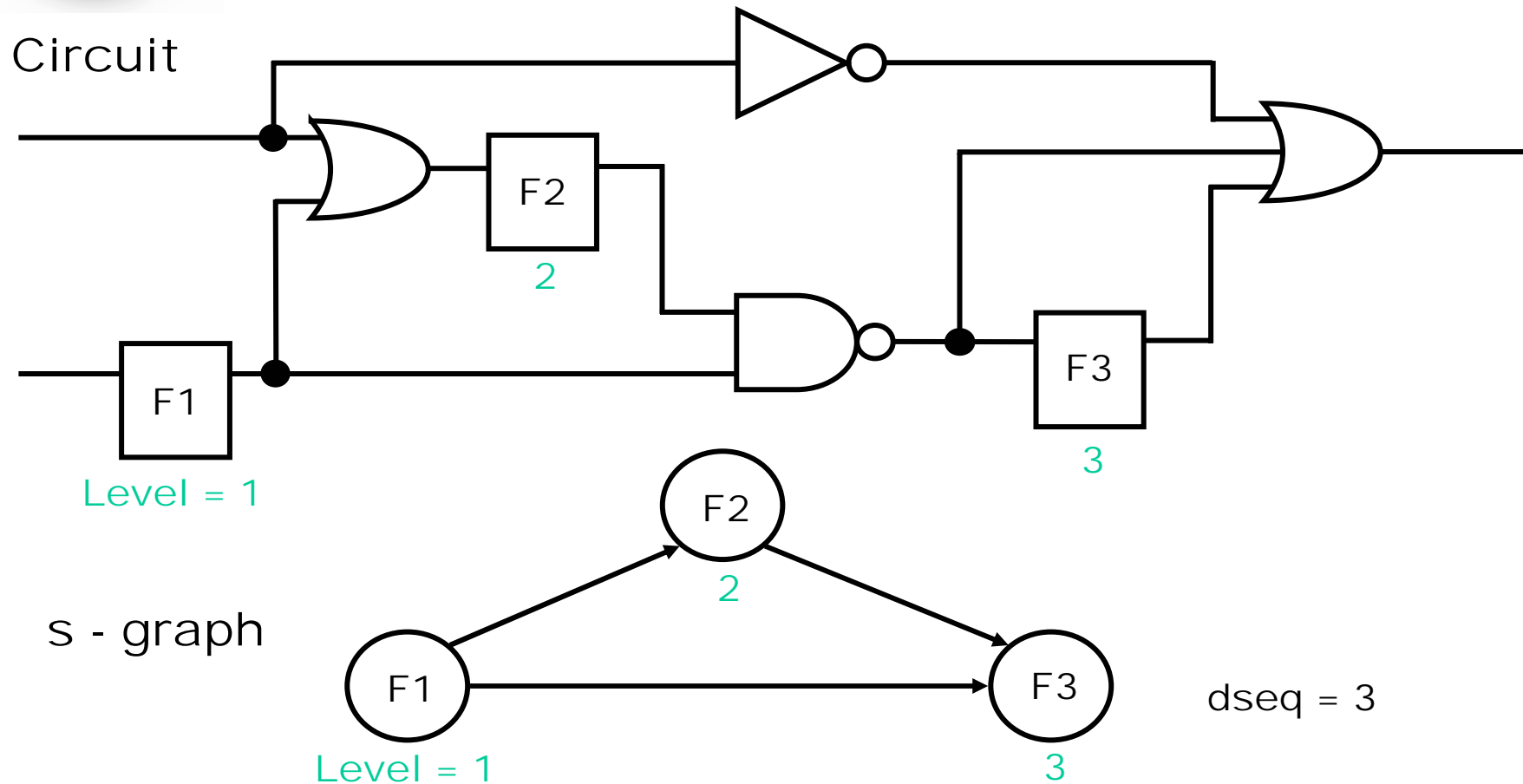


Cycle-Free Circuits

- Characterized by absence of cycles among flip-flops and a sequential depth, $dseq$.
- $dseq$ is the maximum number of flip-flops on any path between PI and PO.
- Both good and faulty circuits are initializable.
- Test sequence length for a fault is bounded by $dseq + 1$.



Cycle-Free Example

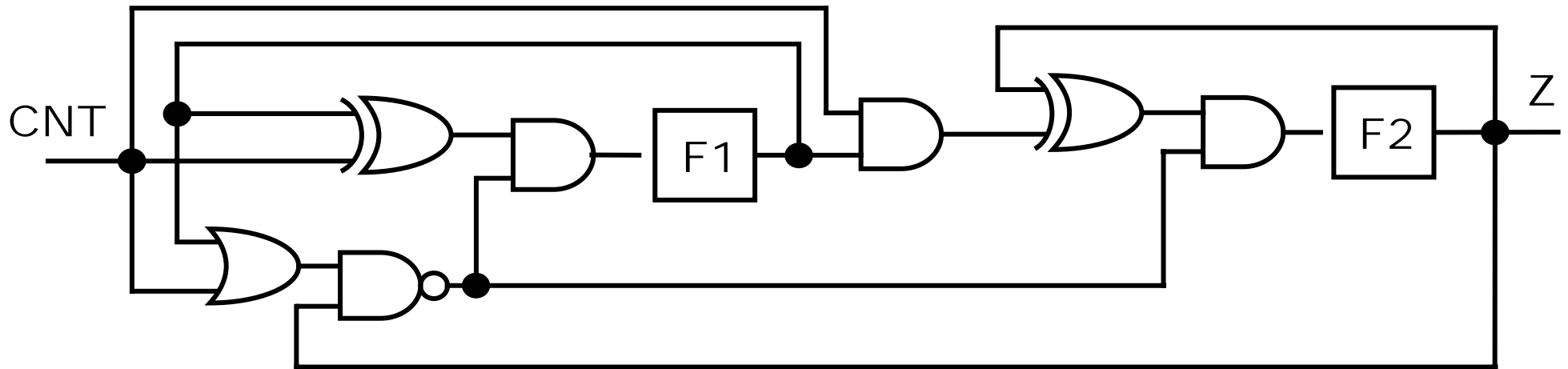


All faults are testable. See Example 8.6.

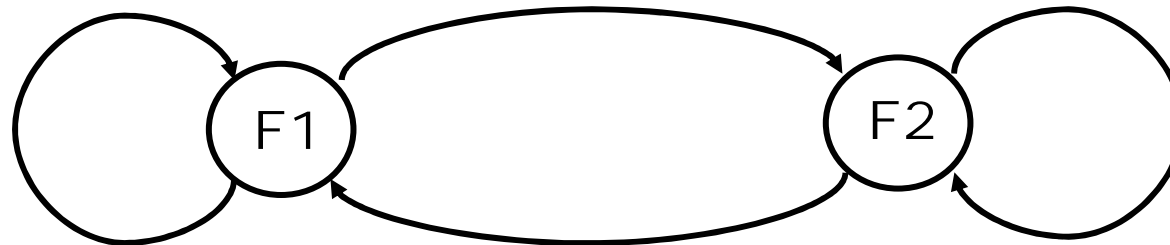


Cyclic Circuit Example

Modulo-3 counter



s - graph





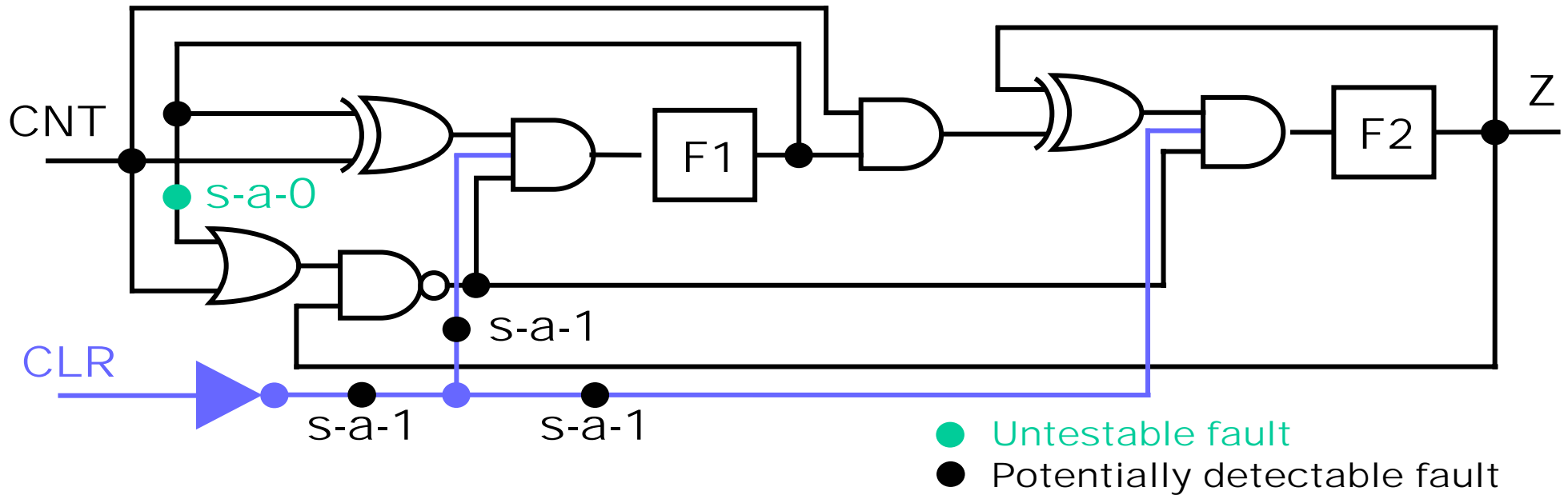
Modulo-3 Counter

- ❑ Cyclic structure - Sequential depth is undefined.
- ❑ Circuit is not initializable. No tests can be generated for any stuck-at fault.
- ❑ After expanding the circuit to $9^{N_{ff}} = 81$, or fewer, time-frames ATPG program calls any given target fault untestable.
- ❑ Circuit can only be functionally tested by multiple observations.
- ❑ Functional tests, when simulated, give no fault coverage.

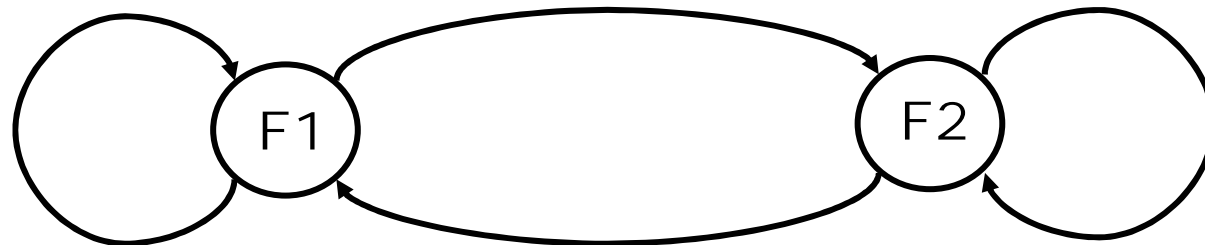


Adding Initializing Hardware

Initializable modulo-3 counter



s - graph





Benchmark Circuits

Circuit	s1196	s1238	s1488	s1494
PI	14	14	8	8
PO	14	14	19	19
FF	18	18	6	6
Gates	529	508	653	647
Structure	Cycle-free	Cycle-free	Cyclic	Cyclic
Seq. depth	4	4	--	--
Total faults	1242	1355	1486	1506
Detected faults	1239	1283	1384	1379
Potentially detected faults	0	0	2	2
Untestable faults	3	72	26	30
Abandoned faults	0	0	76	97
Fault coverage (%)	99.8	94.7	93.1	91.6
Fault efficiency (%)	100.0	100.0	94.8	93.4
Max. sequence length	3	3	24	28
Total test vectors	313	308	525	559
Gentest CPU s (Sparc 2)	10	15	19941	19183



Sequential Circuit ATPG

Simulation-Based Methods

- ❑ Use of fault simulation for test generation
- ❑ Contest
 - Directed search
 - Cost functions
- ❑ *Genetic Algorithms*
- ❑ *Spectral Methods*
- ❑ Summary

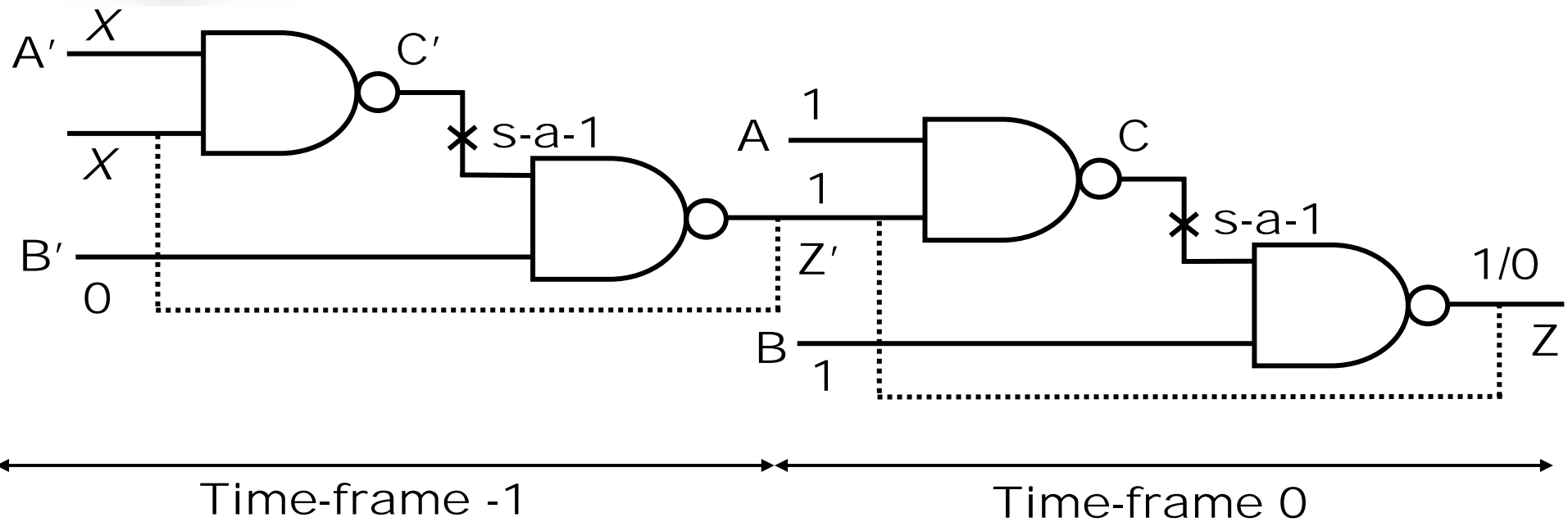


Motivation

- ❑ Difficulties with time-frame method:
 - Long initialization sequence
 - Impossible initialization with three-valued logic
 - Circuit modeling limitations
 - Timing problems - tests can cause races/hazards
 - High complexity
 - Inadequacy for asynchronous circuits
- ❑ Advantages of simulation-based methods
 - Advanced fault simulation technology
 - Accurate simulation model exists for verification
 - Variety of tests - functional, heuristic, random
 - Used since early 1960s



A Test Causing Race



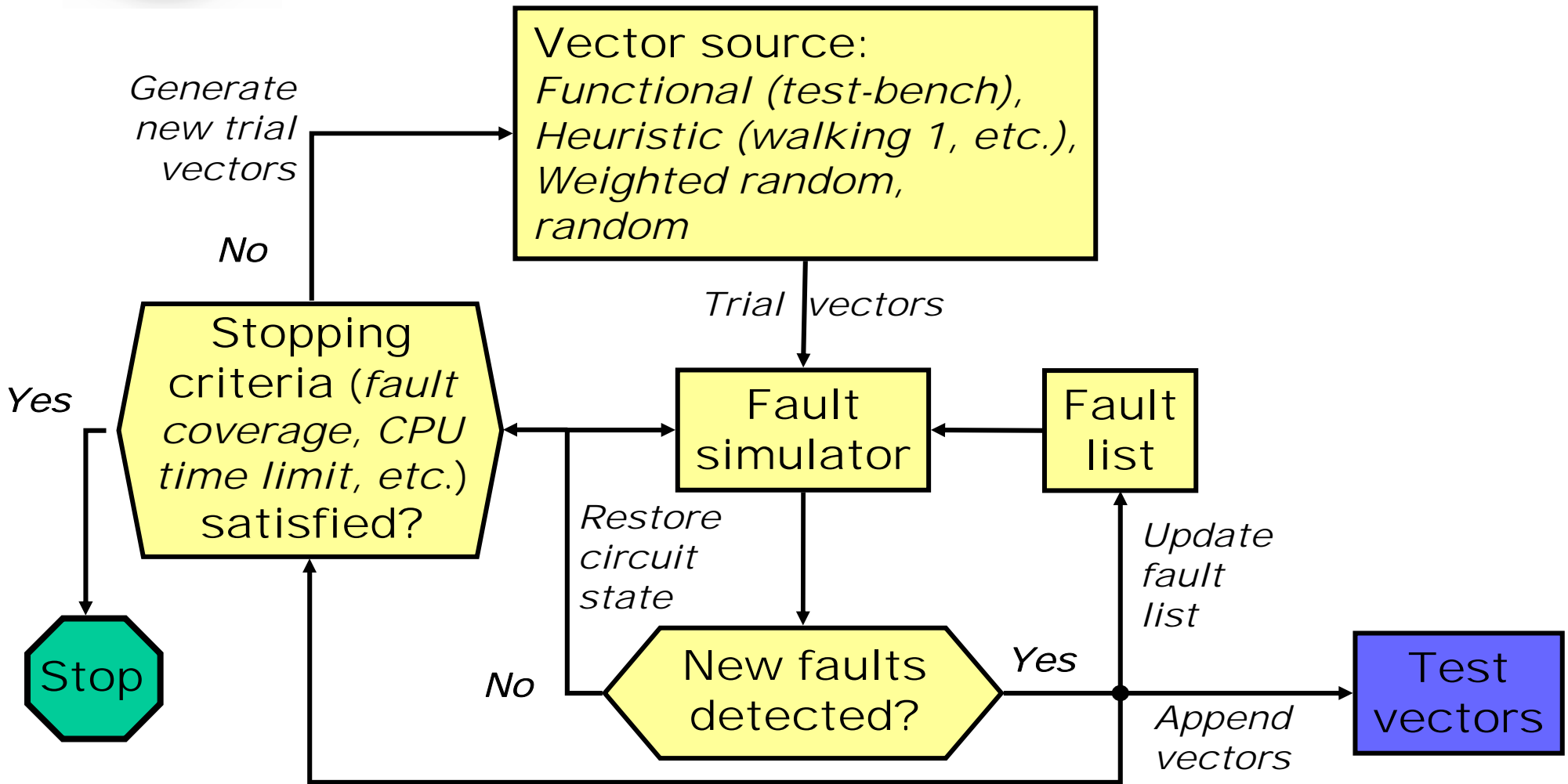
Test is a two-vector sequence: $X0, 11$

$10, 11$ is a good test; no race in fault-free circuit

$00, 11$ causes a race condition in fault-free circuit



Using Fault Simulator





Background

- Seshu and Freeman, 1962, Asynchronous circuits, parallel fault simulator, single-input changes vectors.
- Breuer, 1971, Random sequences, sequential circuits
- Agrawal and Agrawal, 1972, Random vectors followed by D-algorithm, combinational circuits.
- Shuler, *et al.*, 1975, Concurrent fault simulator, random vectors, sequential circuits.
- Parker, 1976, Adaptive random vectors, combinational circuits.
- Agrawal, Cheng and Agrawal, 1989, Directed search with cost-function, concurrent fault simulator, sequential circuits.
- Srinivas and Patnaik, 1993, Genetic algorithms; Saab, *et al.*, 1996; Corno, *et al.*, 1996; Rudnick, *et al.*, 1997; Hsiao, *et al.*, 1997.

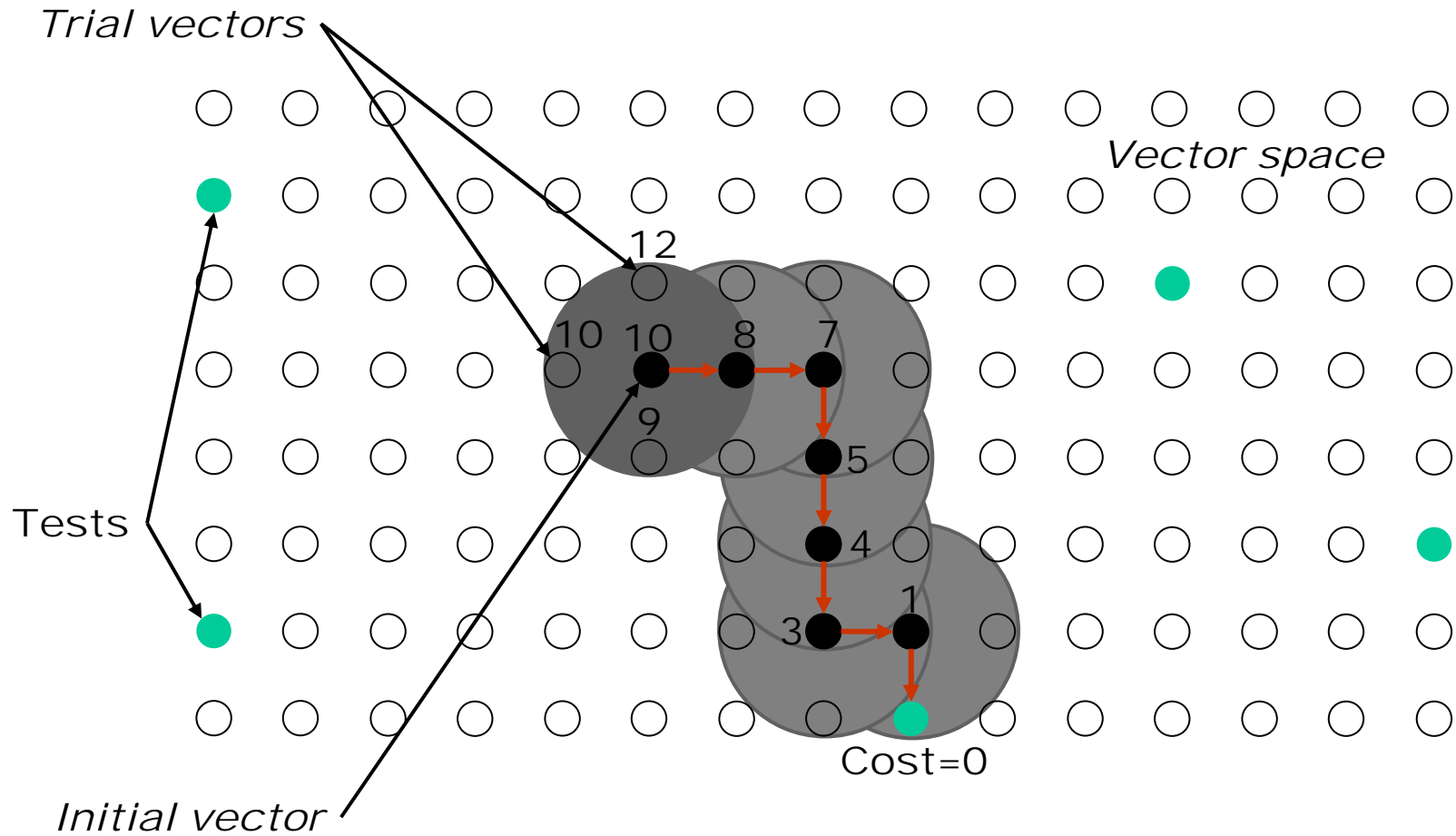


Contest

- ❑ A Concurrent test generator for sequential circuit testing (Contest).
- ❑ Search for tests is guided by cost-functions.
- ❑ Three-phase test generation:
 - Initialization - no faults targeted; cost-function computed by true-value simulator.
 - Concurrent phase - all faults targeted; cost function computed by a concurrent fault simulator.
 - Single fault phase - faults targeted one at a time; cost function computed by true-value simulation and dynamic testability analysis.
- ❑ Ref.: Agrawal, *et al.*, IEEE-TCAD, 1989.



Directed Search





Cost Function

- ❑ Defined for required objective (initialization or fault detection).
- ❑ Numerically grades a vector for suitability to meet the objective.
- ❑ Cost function = 0 for any vector that perfectly meets the objective.
- ❑ Computed for an input vector from true-value or fault simulation.



Phase I: Initialization

- Initialize test sequence with arbitrary, random, or given vector or sequence of vectors.
- Set all flip-flops in *unknown* (X) state.
- Cost function:
 - Cost = Number of flip-flops in the unknown state
 - Cost computed from true-value simulation of trial vectors
- Trial vectors: A heuristically generated vector set from the previous vector(s) in the test sequence; e.g., all vectors at unit Hamming distance from the last vector may form a trial vector set.
- Vector selection: Add the minimum cost trial vector to the test sequence. Repeat trial vector generation and vector selection until cost becomes zero or drops below some given value.

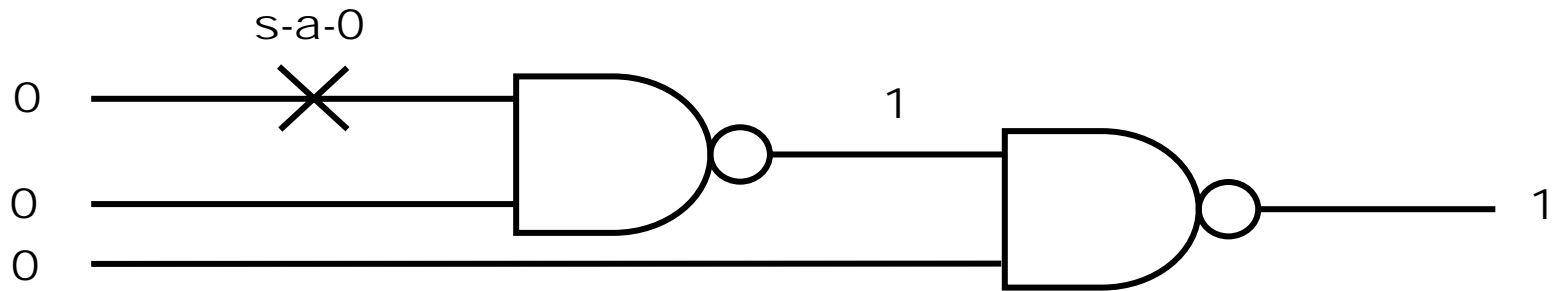


Phase II: Concurrent Fault Detection

- Initially test sequence contains vectors from Phase I.
- Simulate all faults and drop detected faults.
- Compute a *distance cost function* for trial vectors:
 - Simulate all undetected faults for the trial vector.
 - For each fault, find the shortest *fault distance* (in number of gates) between its fault effect and a PO.
 - Cost function is the sum of fault distances for all undetected faults.
- Trial vectors: Generate trial vectors using the unit Hamming distance or any other heuristic.
- Vector selection:
 - Add the trial vector with the minimum distance cost function to test sequence.
 - Remove faults with zero fault distance from the fault list.
 - Repeat trial vector generation and vector selection until fault list is reduced to given size.



Distance Cost Function



Initial vector

0
0
0

∞

Trial vectors

1 0 0
0 1 0
0 0 1

2 ∞ ∞

Trial vectors

0 1 1
0 1 0
0 0 1

∞ 1 2

Trial vectors

0 1 1
1 0 1
0 0 1

∞ 2 0

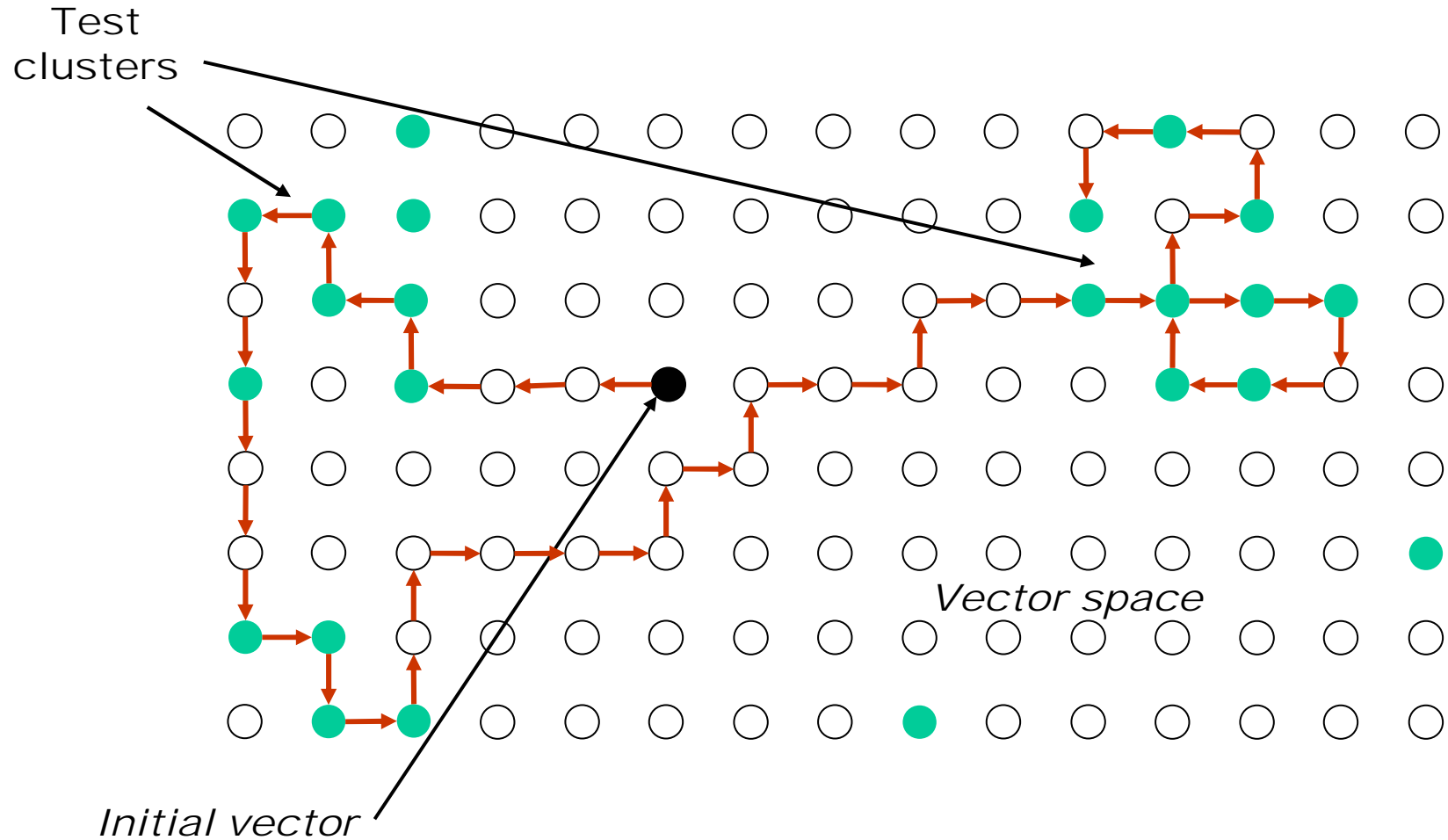
Fault detected

Minimum cost vector

Distance cost function for s-a-0 fault

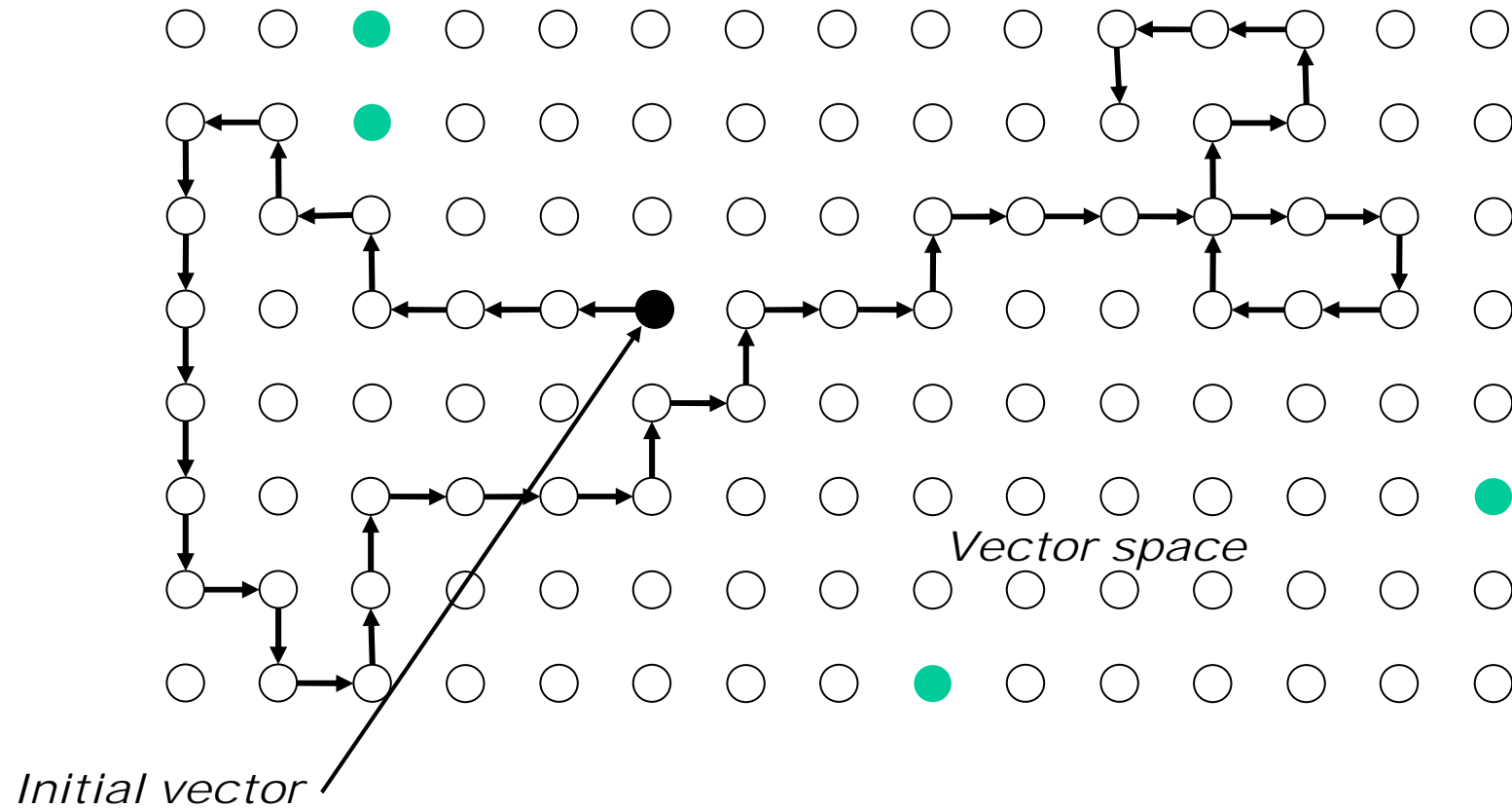


Concurrent Test Generation





Need for Phase III





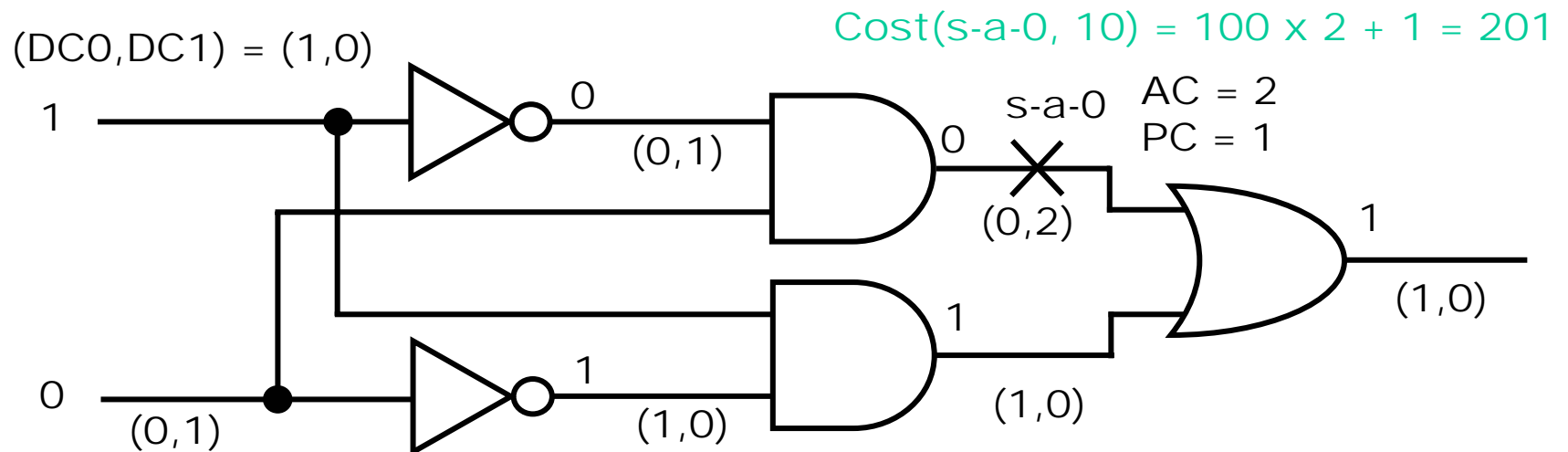
Phase III: Single Fault Target

- Cost (fault, input vector) = $K \times AC + PC$
 - *Activation cost* (AC) is the dynamic controllability of the faulty line.
 - *Propagation cost* (PC) is the minimum (over all paths to POs) dynamic observability of the faulty line.
 - K is a large weighting factor, e.g., $K = 100$.
 - Dynamic testability measures (controllability and observability) are specific to the present signal values in the circuit.
 - Cost of a vector is computed for a fault from true-value simulation result.
 - Cost = 0 means fault is detected.
- Trial vector generation and vector selection are similar to other phases.



Dynamic Test. Measures

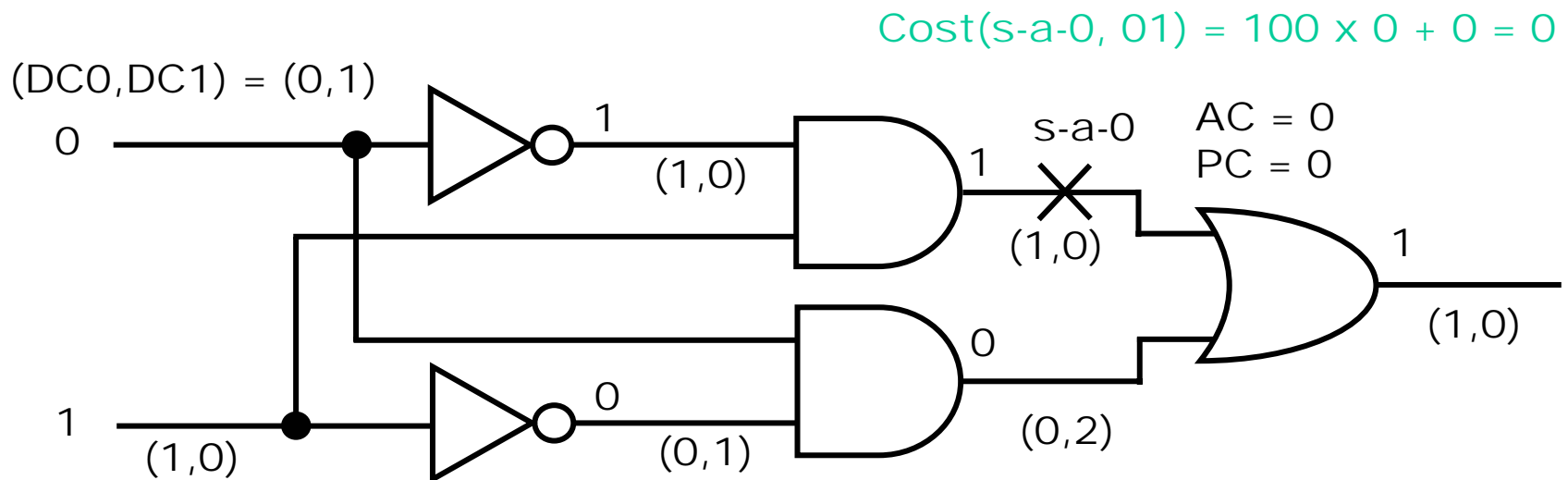
- Number of inputs to be changed to achieve an objective:
 - DC0, DC1 - cost of setting line to 0, 1
 - AC = DC0 (or DC1) at fault site for s-a-1 (or s-a-0)
 - PC - cost of observing line
- Example: A vector with non-zero cost.





Dynamic Test. Measures

- Example: A vector (test) with zero cost.





Other Features

- More on dynamic testability measures:
 - Unknown state - A signal can have three states.
 - Flip-flops - Output DC is input DC, plus a large constant (say, 100), to account for time frames.
 - Fanout - PC for stem is minimum of branch PCs.
- Types of circuits: Tests are generated for any circuit that can be simulated:
 - Combinational - No clock; single vector tests.
 - Asynchronous - No clock; simulator analyzes hazards and oscillations, 3-states, test sequences.
 - Synchronous - Clocks specified, flip-flops treated as black-boxes, 3-states, implicit-clock test sequences.



Contest Result: s5378

- 35 PIs, 49 POs, 179 FFs, 4,603 faults.
- Synchronous, single clock.

	Contest	Random vectors	Gentest**
Fault coverage	75.5%	67.6%	72.6%
Untestable faults	0	0	122
Test vectors	1,722	57,532	490
Trial vectors used	57,532	--	--
Test gen. CPU time#	3 min.*	0	4.5 hrs.
Fault sim. CPU time#	9 min.*	9 min.	10 sec.

Sun Ultra II, 200MHz CPU

*Estimated time

**Time-frame expansion (higher coverage possible with more CPU time)



Genetic Algorithms (GAs)

- Theory of evolution by natural selection (Darwin, 1809-82.)
 - C. R. Darwin, *On the Origin of Species by Means of Natural Selection*, London: John Murray, 1859.
 - J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.
 - D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, Massachusetts: Addison-Wesley, 1989.
 - P. Mazumder and E. M. Rudnick, *Genetic Algorithms for VLSI Design, Layout and Test Automation*, Upper Saddle River, New Jersey, Prentice Hall PTR, 1999.

- Basic Idea: Population *improves* with each generation.
 - Population
 - Fitness criteria
 - Regeneration rules



GAs for Test Generation

- Population: A set of input vectors or vector sequences.
- Fitness function: Quantitative measures of population succeeding in tasks like initialization and fault detection (reciprocal to cost functions.)
- Regeneration rules (heuristics): Members with higher fitness function values are *selected* to produce new members via transformations like *mutation* and *crossover*.



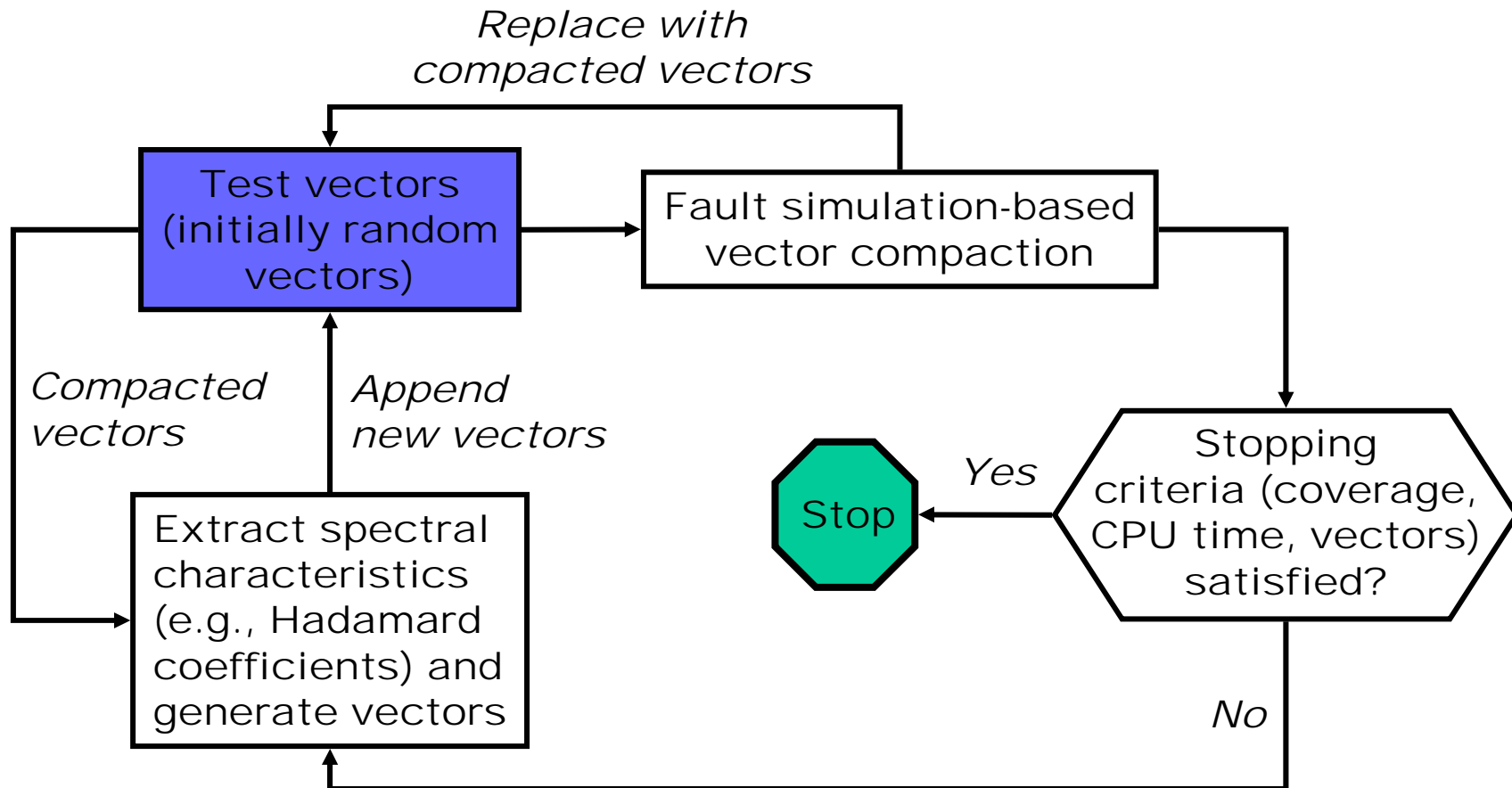
Strategate Results

	s1423	s5378	s35932
Total faults	1,515	4,603	39,094
Detected faults	1,414	3,639	35,100
Fault coverage	93.3%	79.1%	89.8%
Test vectors	3,943	11,571	257
CPU time	1.3 hrs.	37.8 hrs.	10.2 hrs.
HP J200 256MB			

Ref.: M. S. Hsiao, E. M. Rudnick and J. H. Patel, "Dynamic State Traversal for Sequential Circuit Test Generation," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 3, July 2000.



Spectral Methods





Spectral Information

- Random inputs resemble noise and have low coverage of faults.
- Sequential circuit tests are not random:
 - Some PIs are correlated.
 - Some PIs are periodic.
 - Correlation and periodicity can be represented by spectral components, e.g., Hadamard coefficients.
- Vector compaction removes unnecessary vectors without reducing fault coverage:
 - Reverse simulation for combinational circuits (Example 5.5.)
 - Vector restoration for sequential circuits.
- Compaction is similar to noise removal (filtering) and enhances spectral characteristics.



Spectral Method: s5378

	Simulation-based methods			Time-frame expansion	
	Spectral-method*	Strategate	Contest	Hitec	Gentest
Fault cov.	79.14%	79.06%	75.54%	70.19%	72.58%
Vectors	734	11,571	1,722	912	490
CPU time	43.5 min.	37.8 hrs.	12.0 min.	18.4 hrs.	5.0 hrs.
Platform	Ultra Sparc 10	Ultra Sparc 1	Ultra II	HP9000/J200	Ultra II

* A. Giani, S. Sheng, M. S. Hsiao and V. D. Agrawal, "Efficient Spectral Techniques for Sequential ATPG," *Proc. IEEE Design and Test in Europe (DATE) Conf.*, March 2001.



Summary

- ❑ Fault simulation is an effective tool for sequential circuit ATPG.
- ❑ Tests can be generated for any circuit that can be simulated. Timing considerations allow dealing with asynchronous circuits.
- ❑ Simulation-based methods generate better tests but produce more vectors, which can be reduced by compaction.
- ❑ A simulation-based method cannot identify untestable faults.
- ❑ Spectral methods hold potential.