Automatic Test Pattern Generation - III

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 – 13 (Jan 31, 2012)

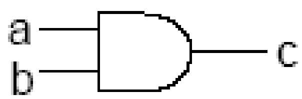
ATPG - Algorithmic

- 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

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

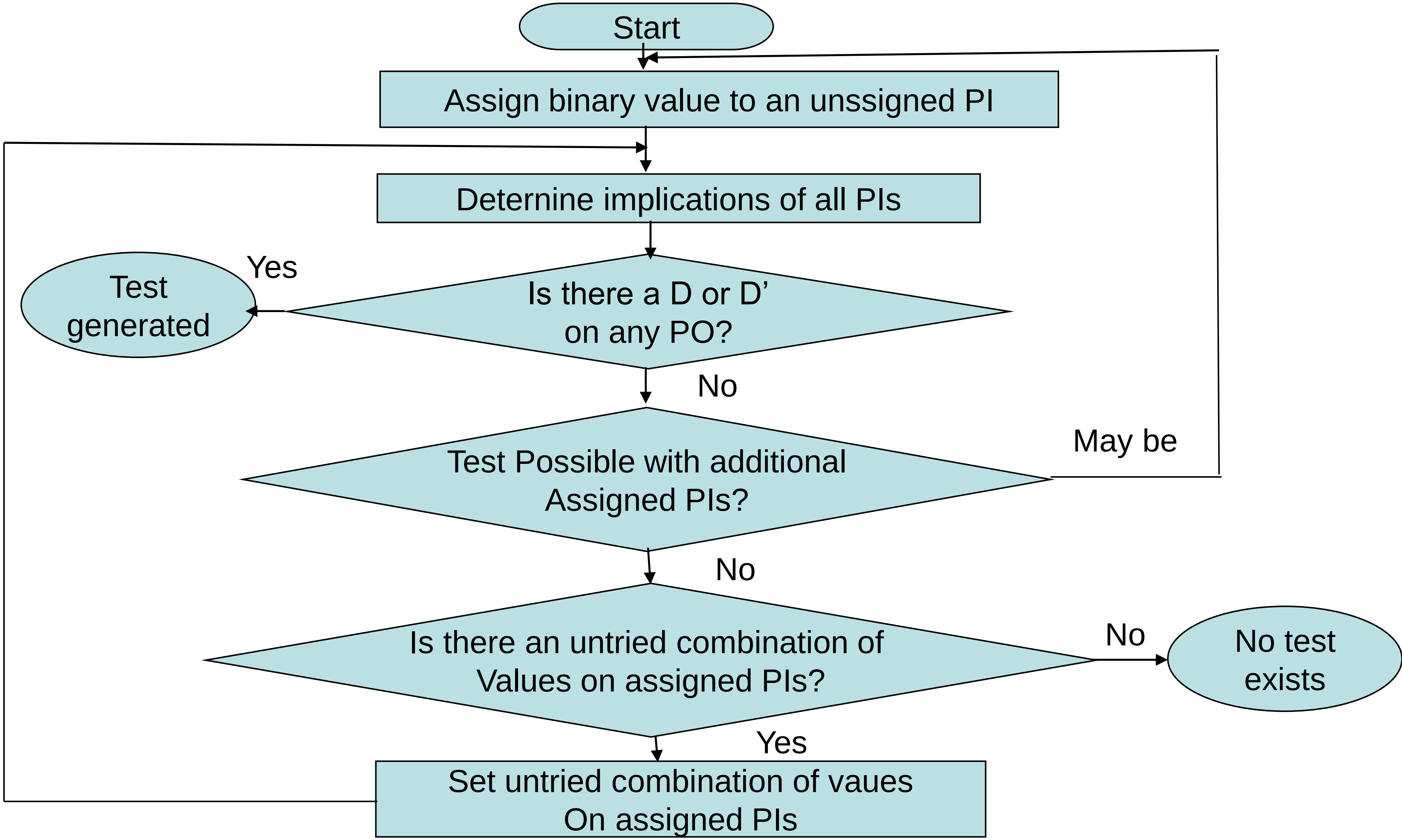
PODEM

- **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

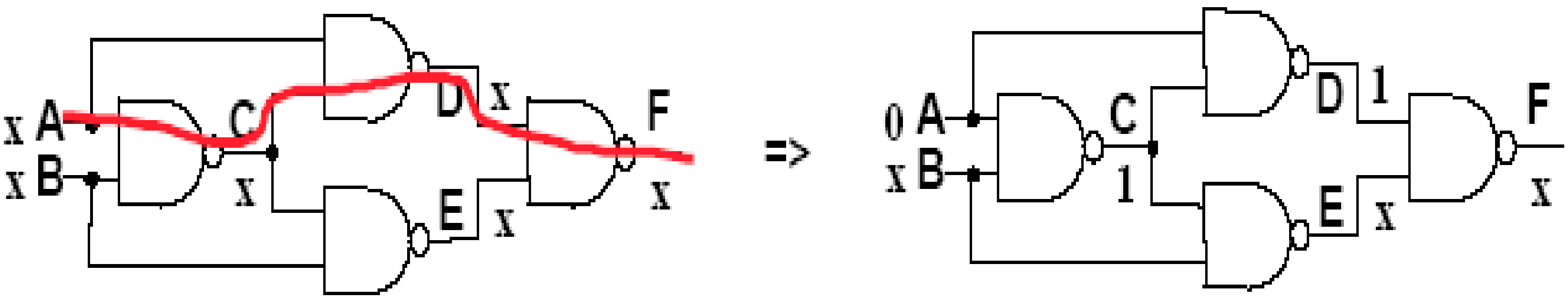
1. **Assign binary value to unassigned PI**
2. **Determine implications of all PIs**
3. **Test Generated? If so, done.**
4. **Test possible with more assigned PIs? If maybe, go to Step 1**
5. **Is there untried combination of values on assigned PIs? If not, exit: untestable fault**
6. **Set untried combination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

PODEM-Algorithm

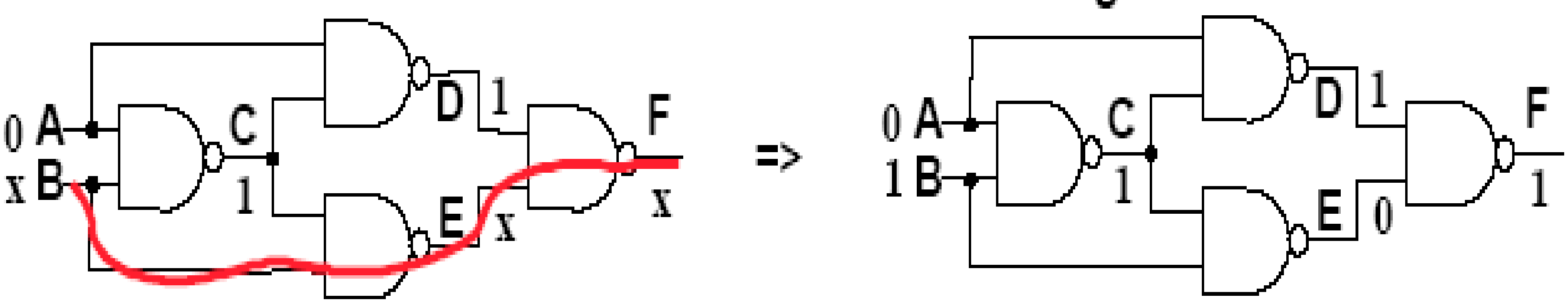


PODEM

Ex: Objective = (F,1).

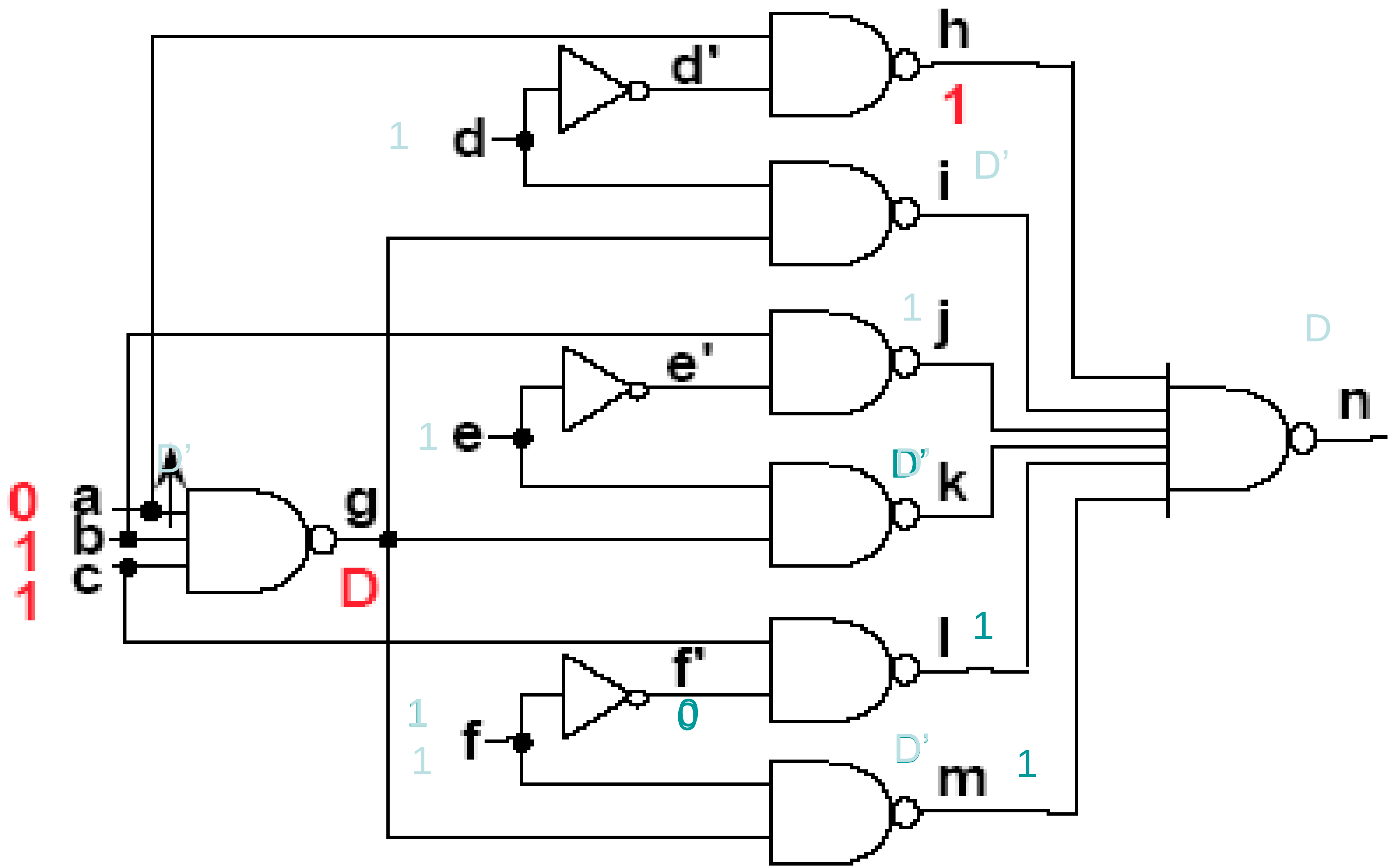


The first time of backtracing

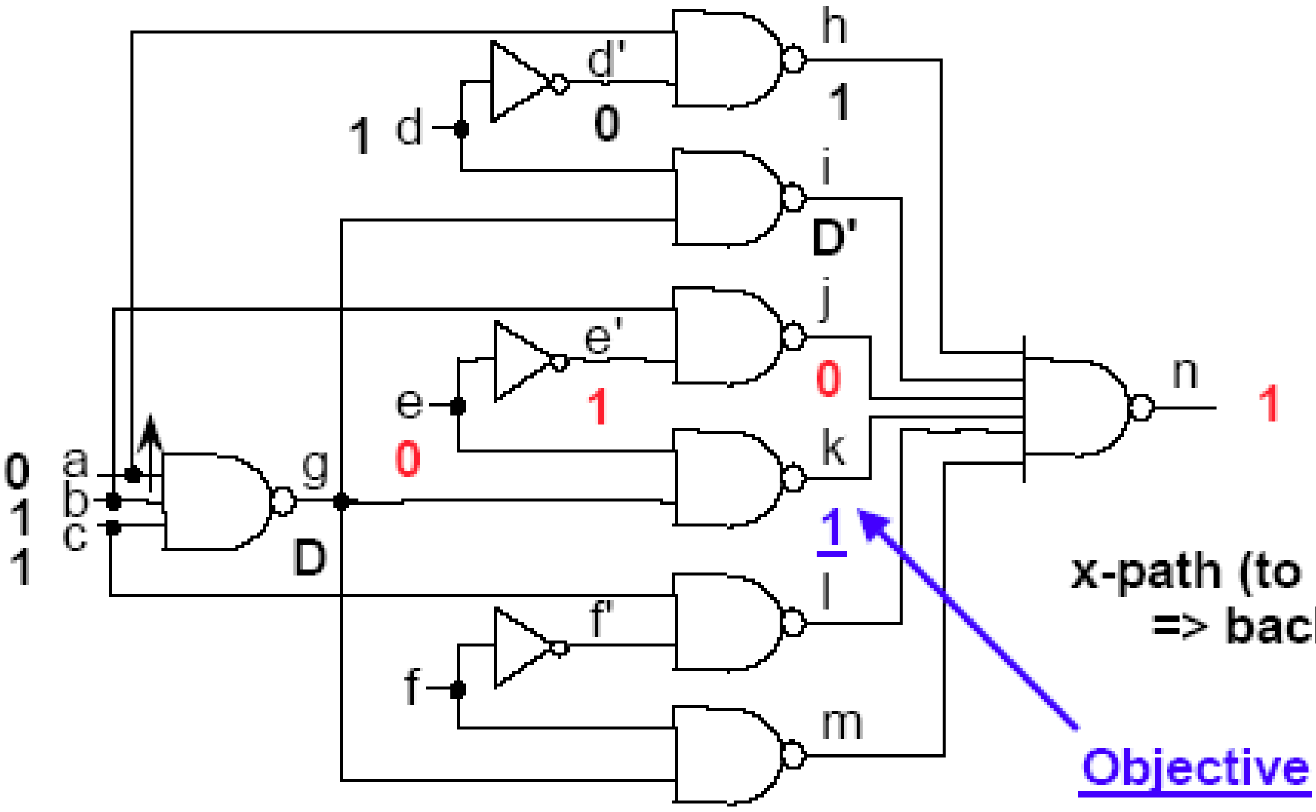


The second time of backtracing

D-Algorithm : Example



PODEM : Example

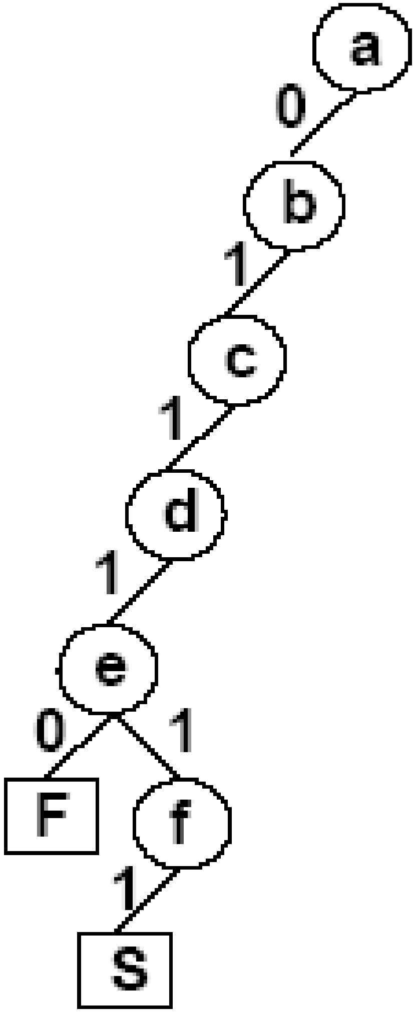


x-path (to PO) check fail => backtracking!!

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



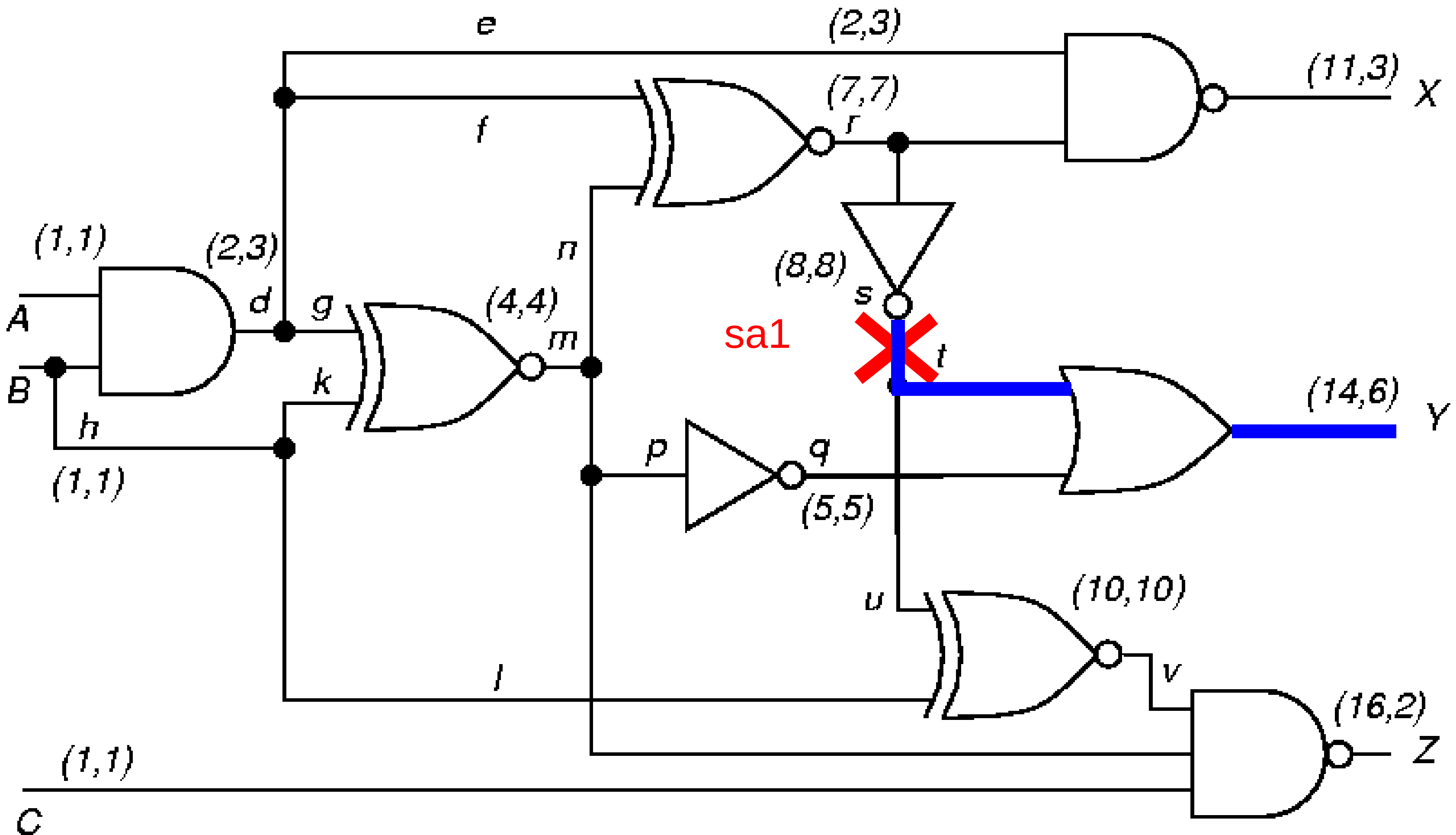
PODEM

PODEM doesn't need

- **Consistency check – conflict can never occur**
- **J-frontier – there are no values that require justification**
- **Backward implication – values are propagated only in forward directions**

Example

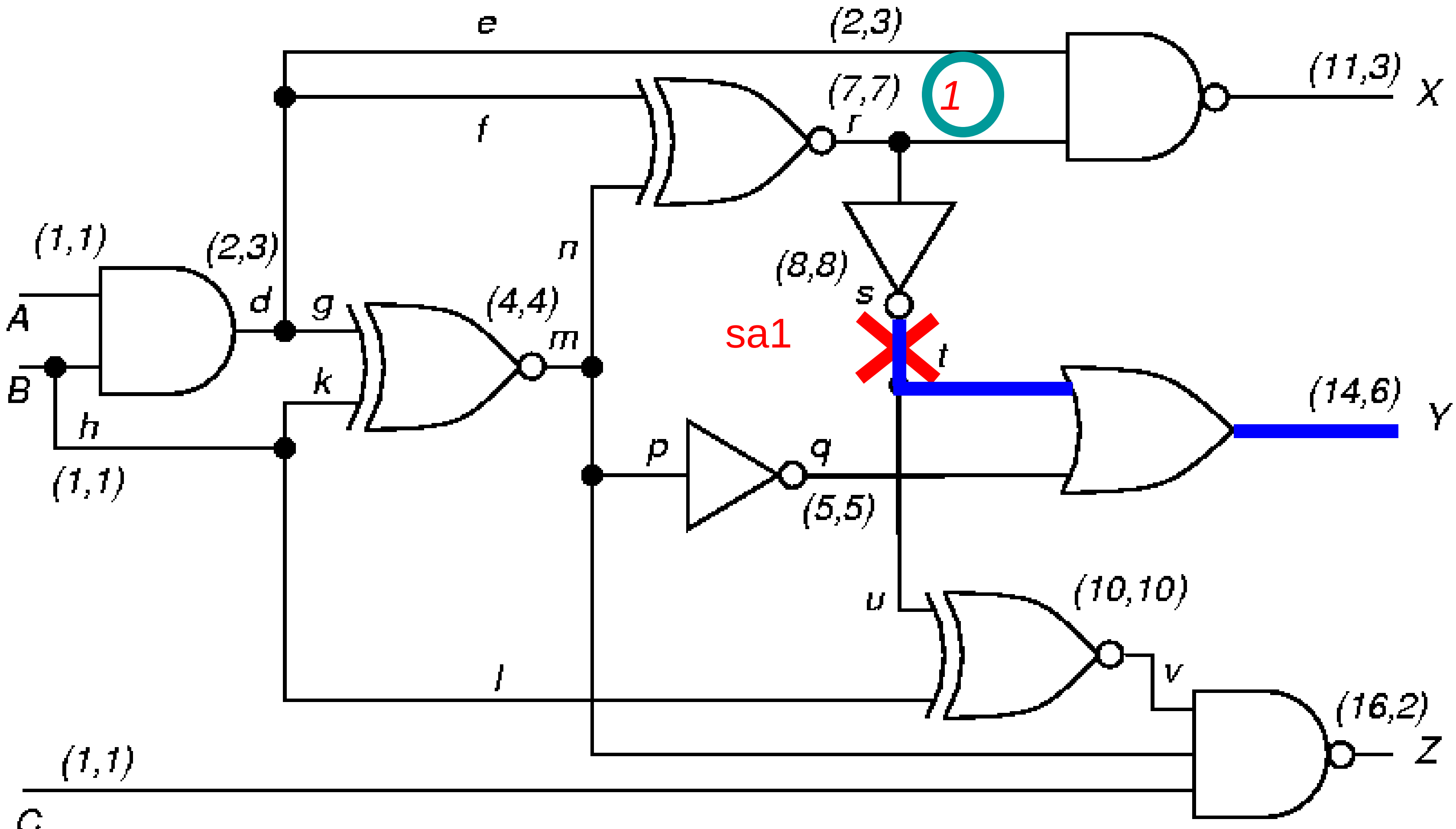
- **Select path *s – Y* for fault propagation**



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

Example -- Step 2 s sa1

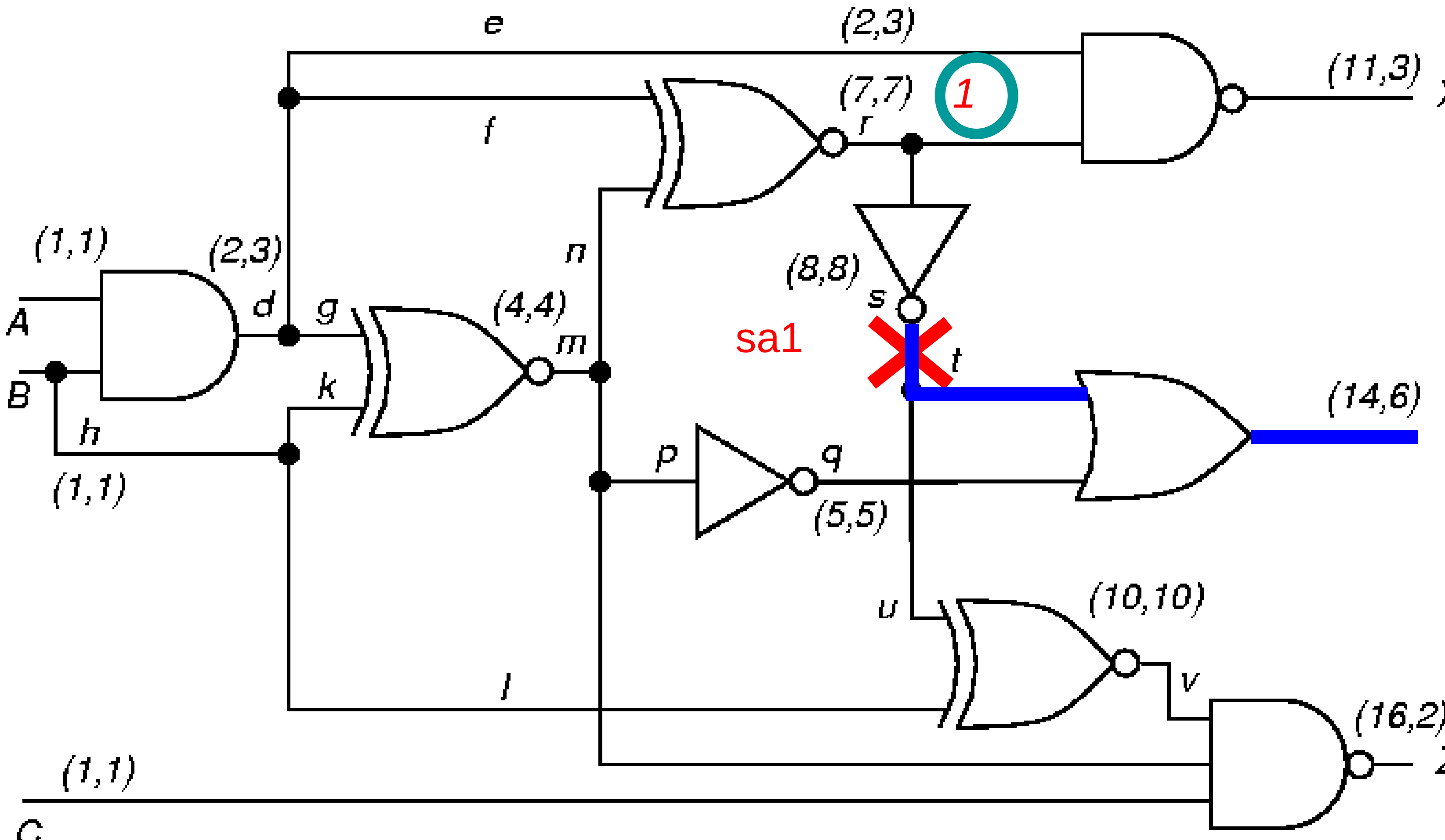
- **Initial objective: Set *r* to 1 to sensitize fault**



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

Example -- Step 3 s sa1

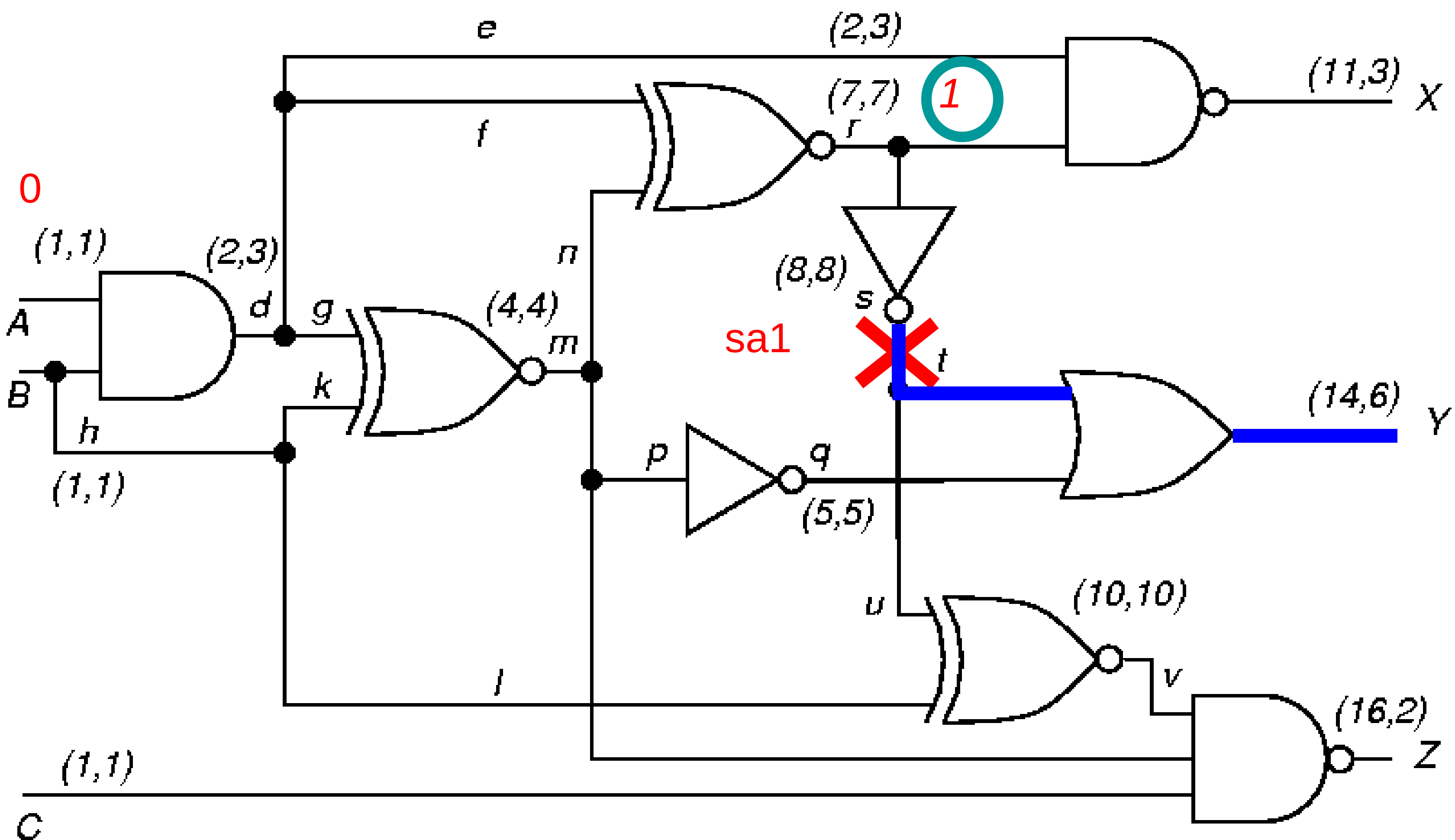
- **Backtrace from *r***



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

Example -- Step 4 s sa1

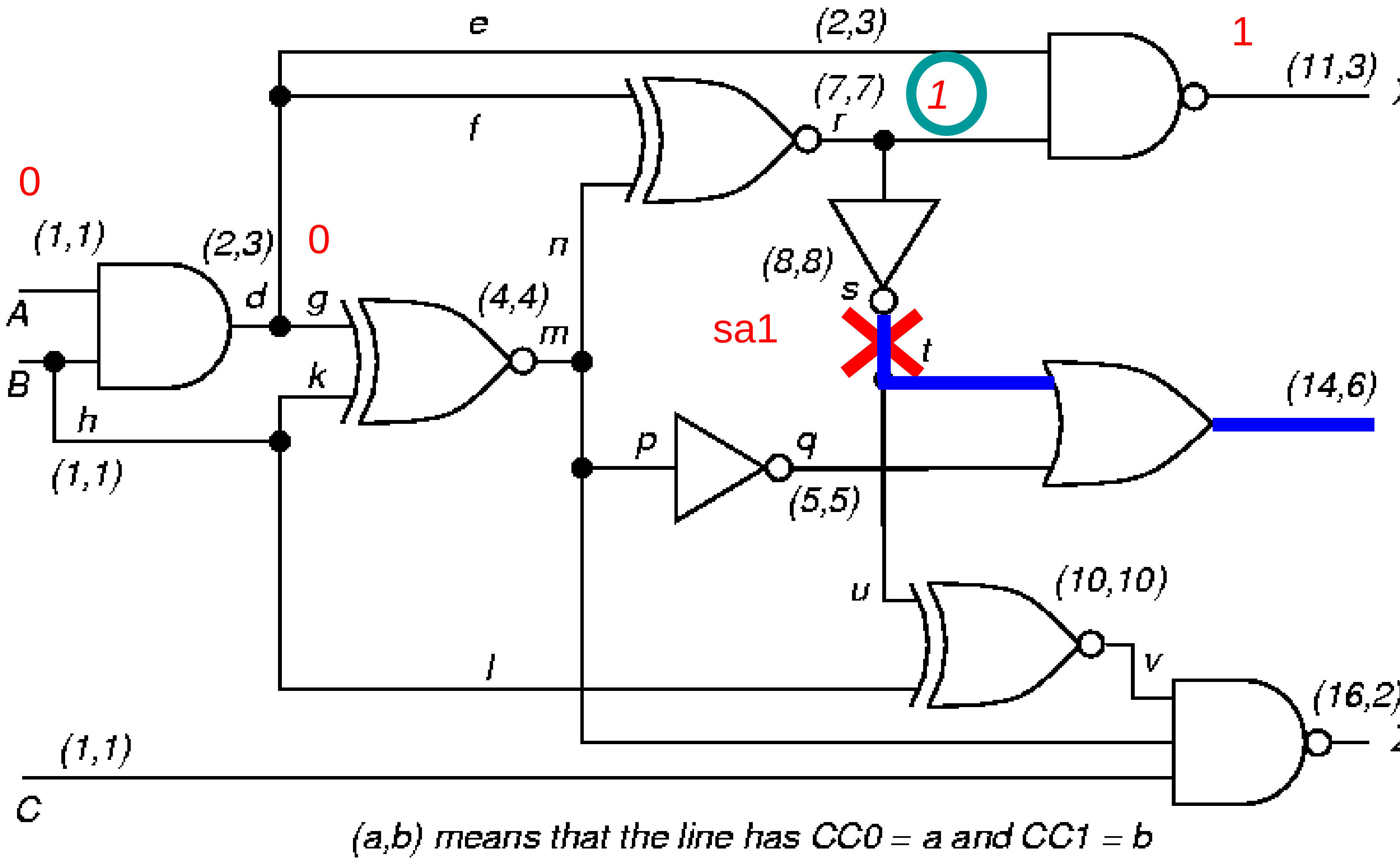
- **Set *A* = 0 in implication stack**



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

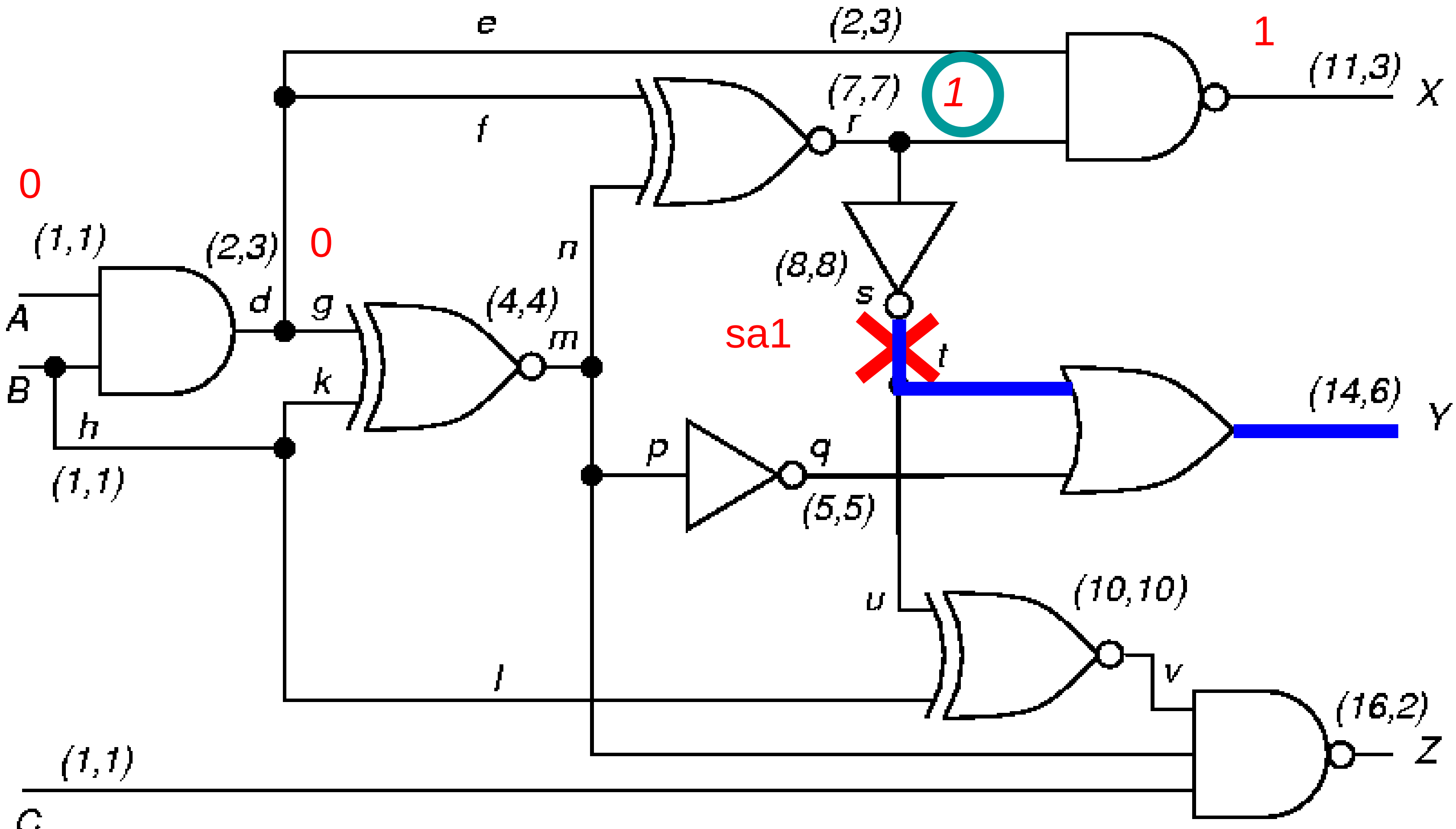
Example -- Step 5 s sa1

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



Example -- Step 6 s sa1

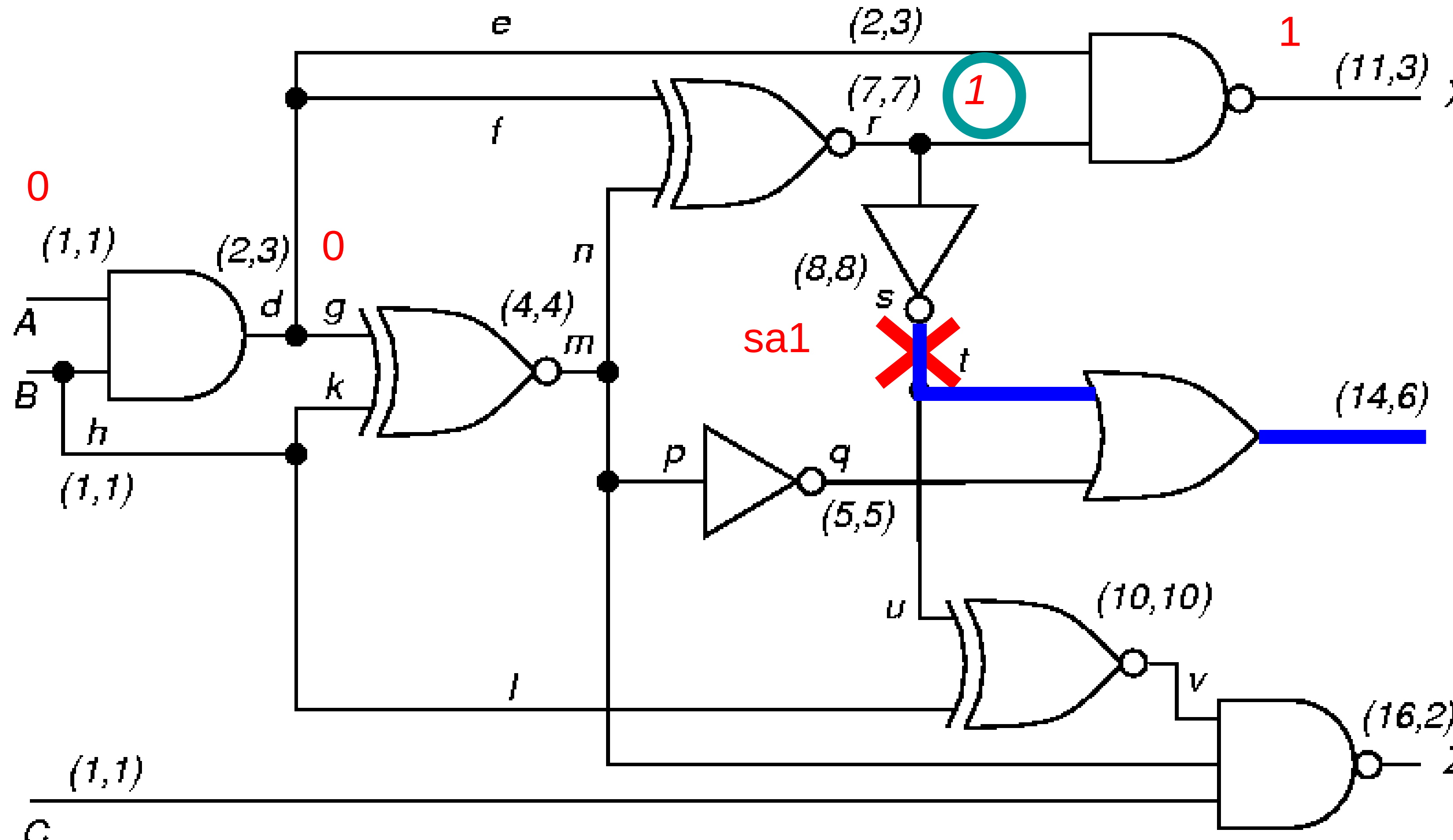
- **Initial objective: set *r* to 1**



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

Example -- Step 7 s sa1

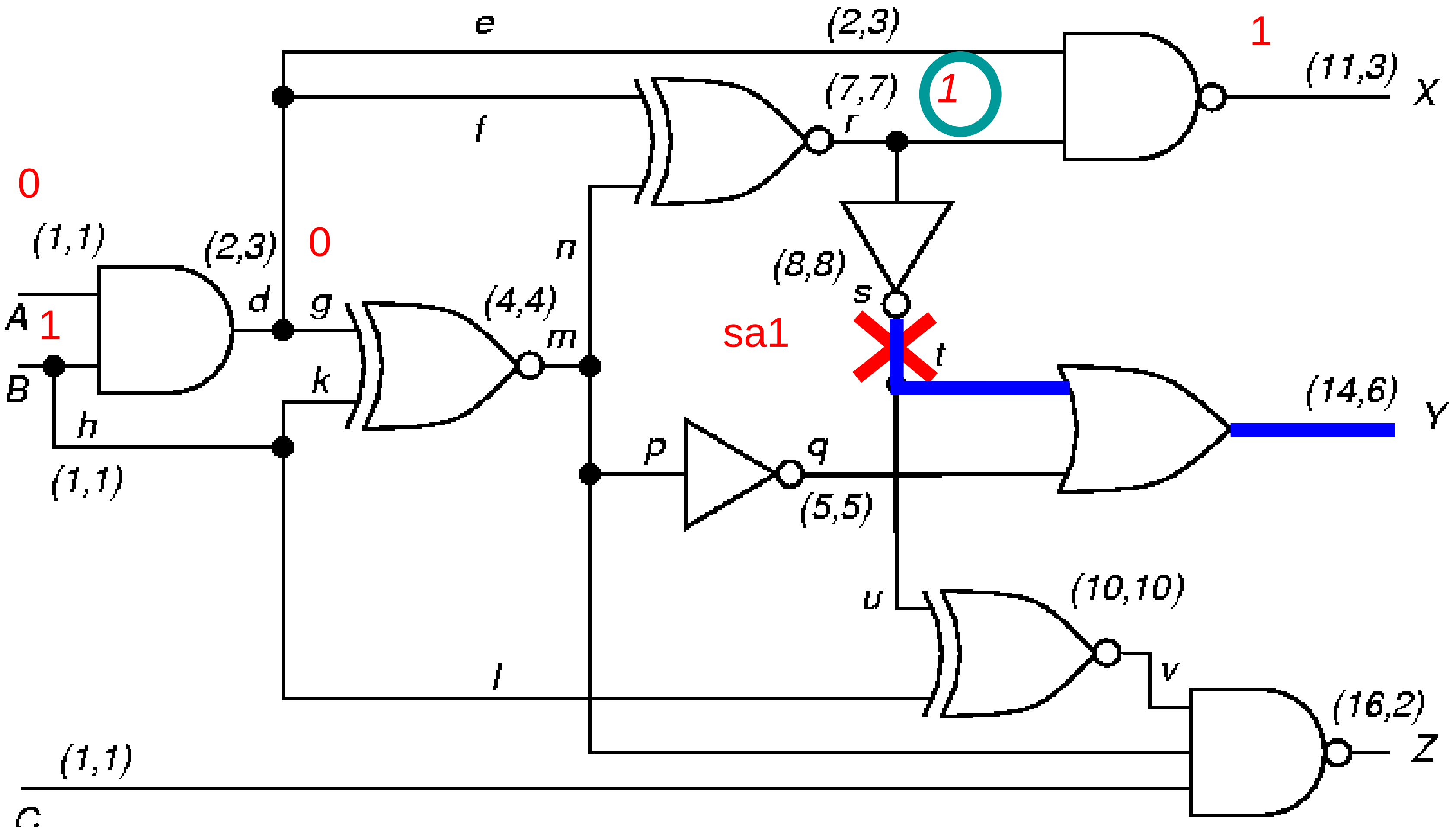
- **Backtrace from *r* again**



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

Example -- Step 8 s sa1

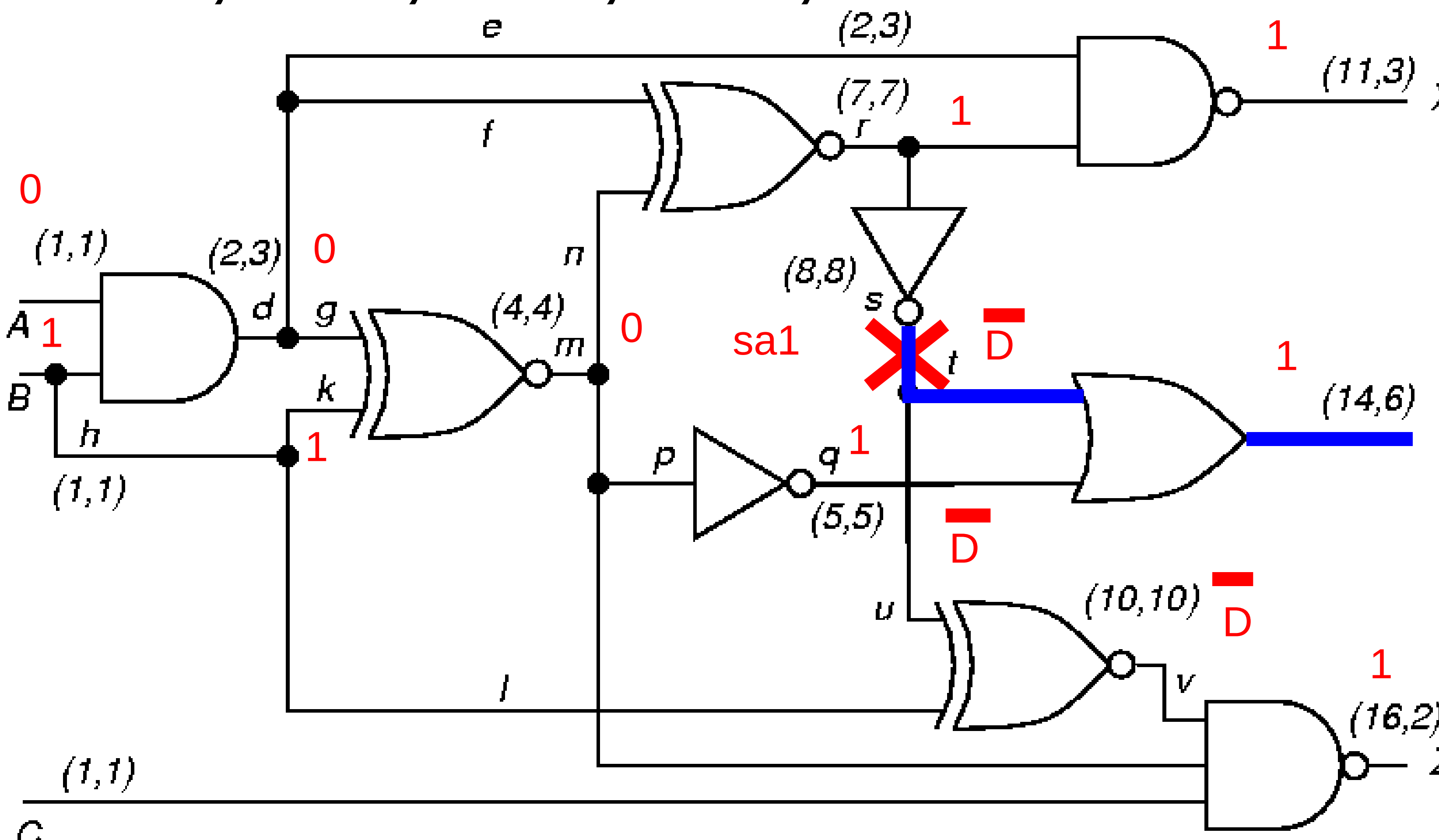
- **Set *B* to 1. Implications in stack: *A* = 0, *B* = 1**



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

Example -- Step 9 s sa1

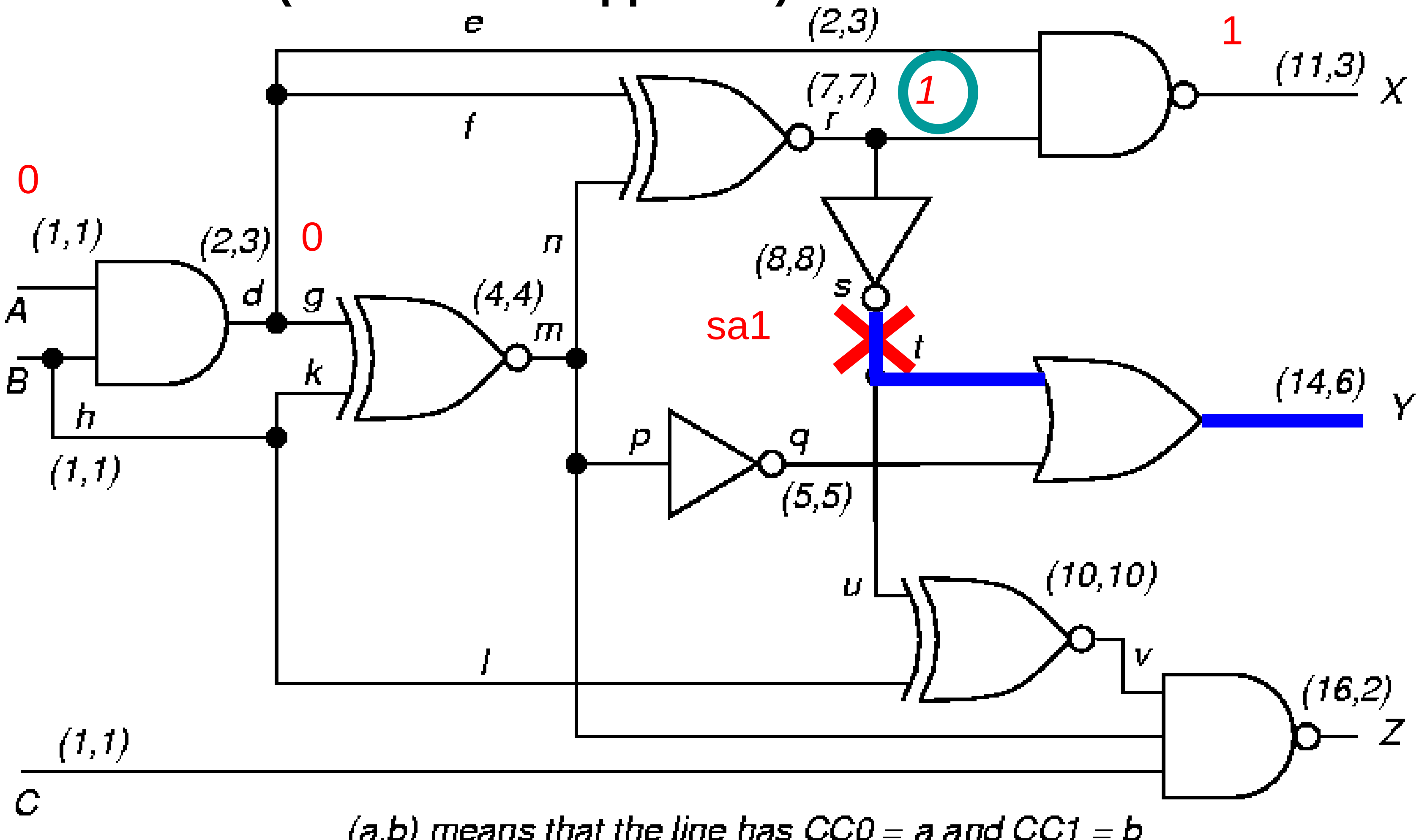
- **Forward implications: *k* = 1, *m* = 0, *r* = 1, *q* = 1, *Y* = 1, *s* = $\overline{\text{D}}$, *u* = $\overline{\text{D}}$, *v* = $\overline{\text{D}}$, *Z* = 1**



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

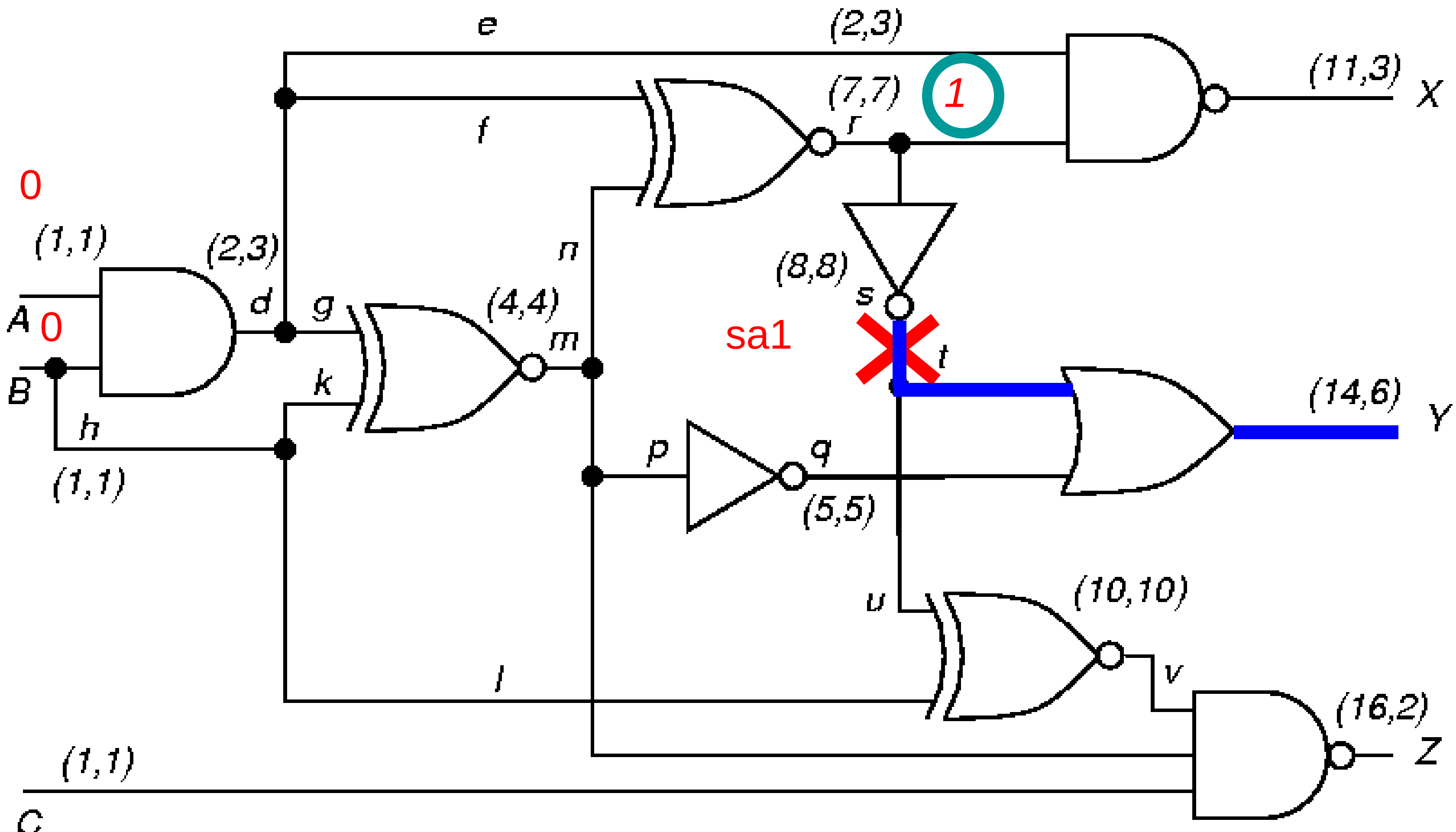
Backtrack -- Step 10 s sa1

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



Step 11 -- s sa1

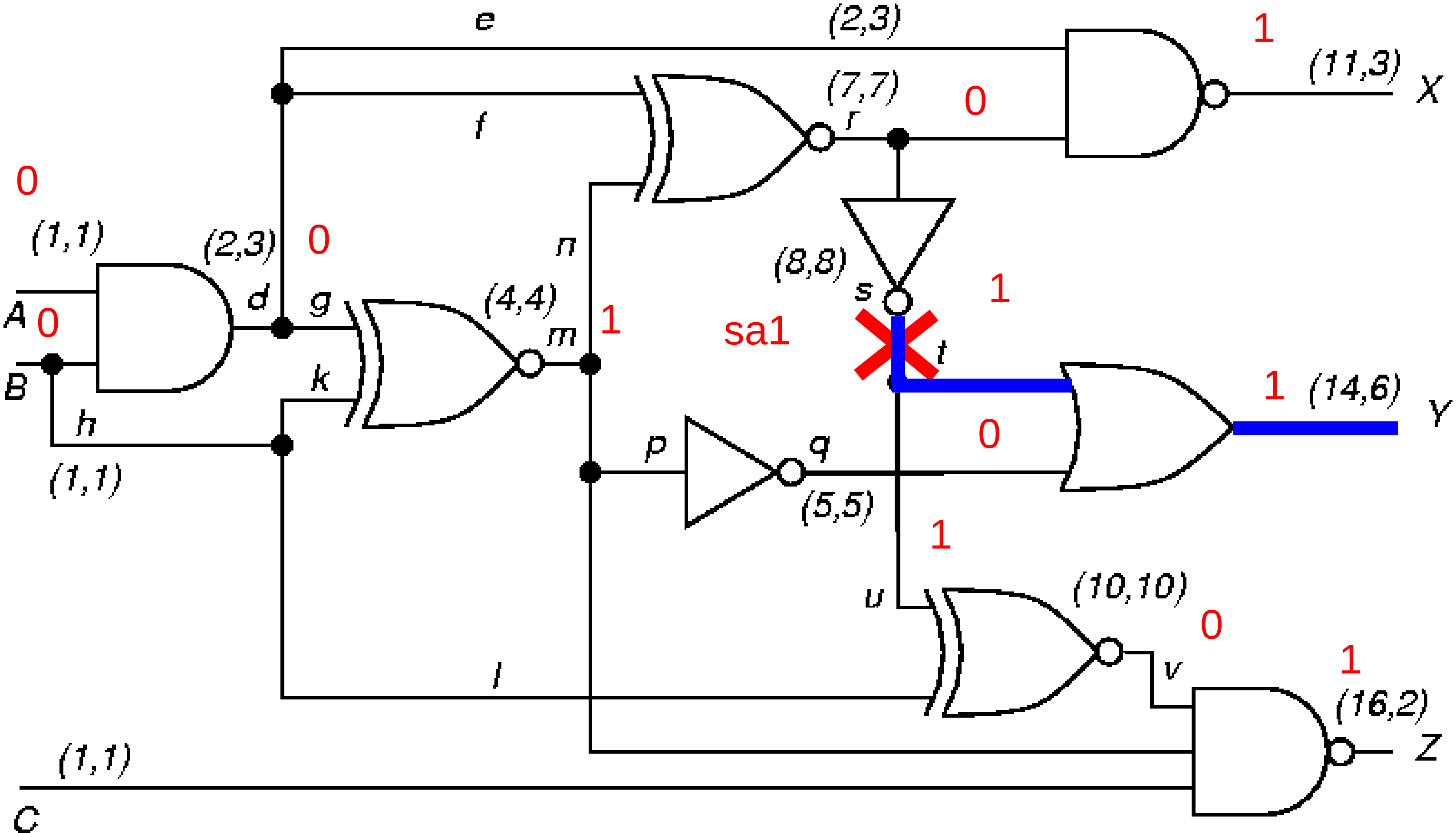
- **Set *B* = 0 (alternate assignment)**



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

Backtrack -- *s* sa1

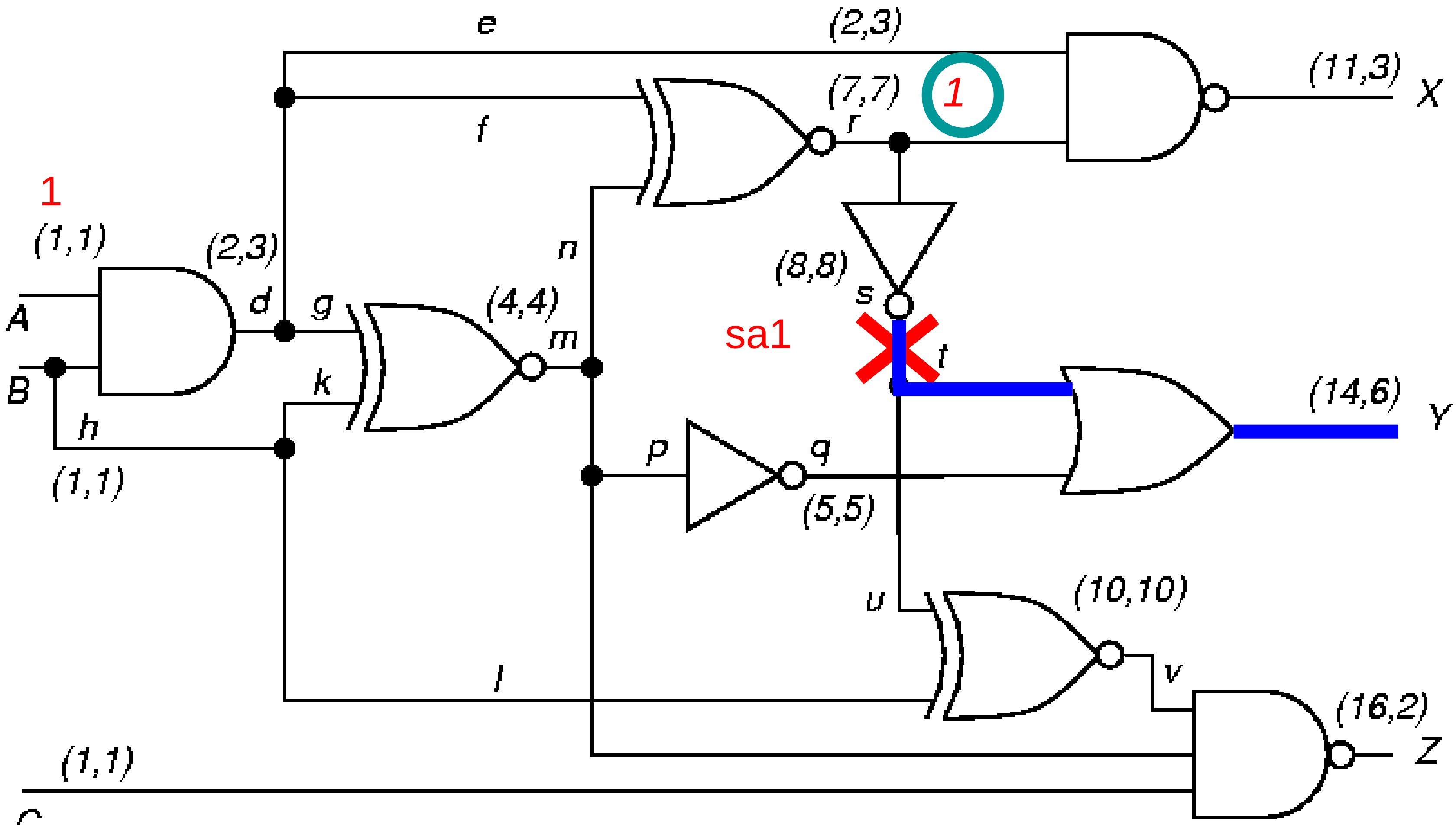
- **Forward implications: *d* = 0, *X* = 1, *m* = 1, *r* = 0, *s* = 1, *q* = 0, *Y* = 1, *v* = 0, *Z* = 1. Fault not sensitized.**



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

Step 13 -- s sa1

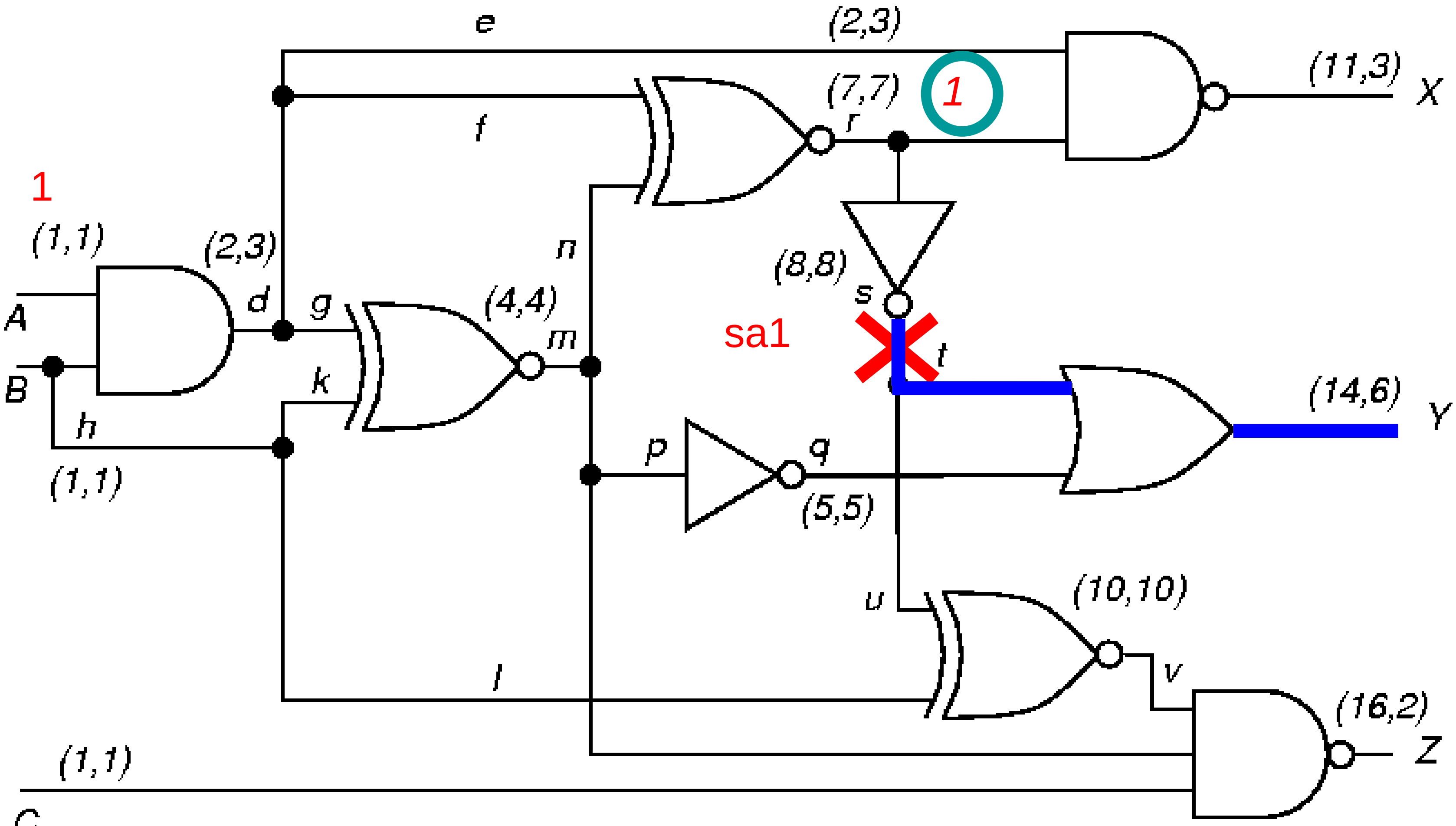
- **Set *A* = 1 (alternate assignment)**



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

Step 14 -- s sa1

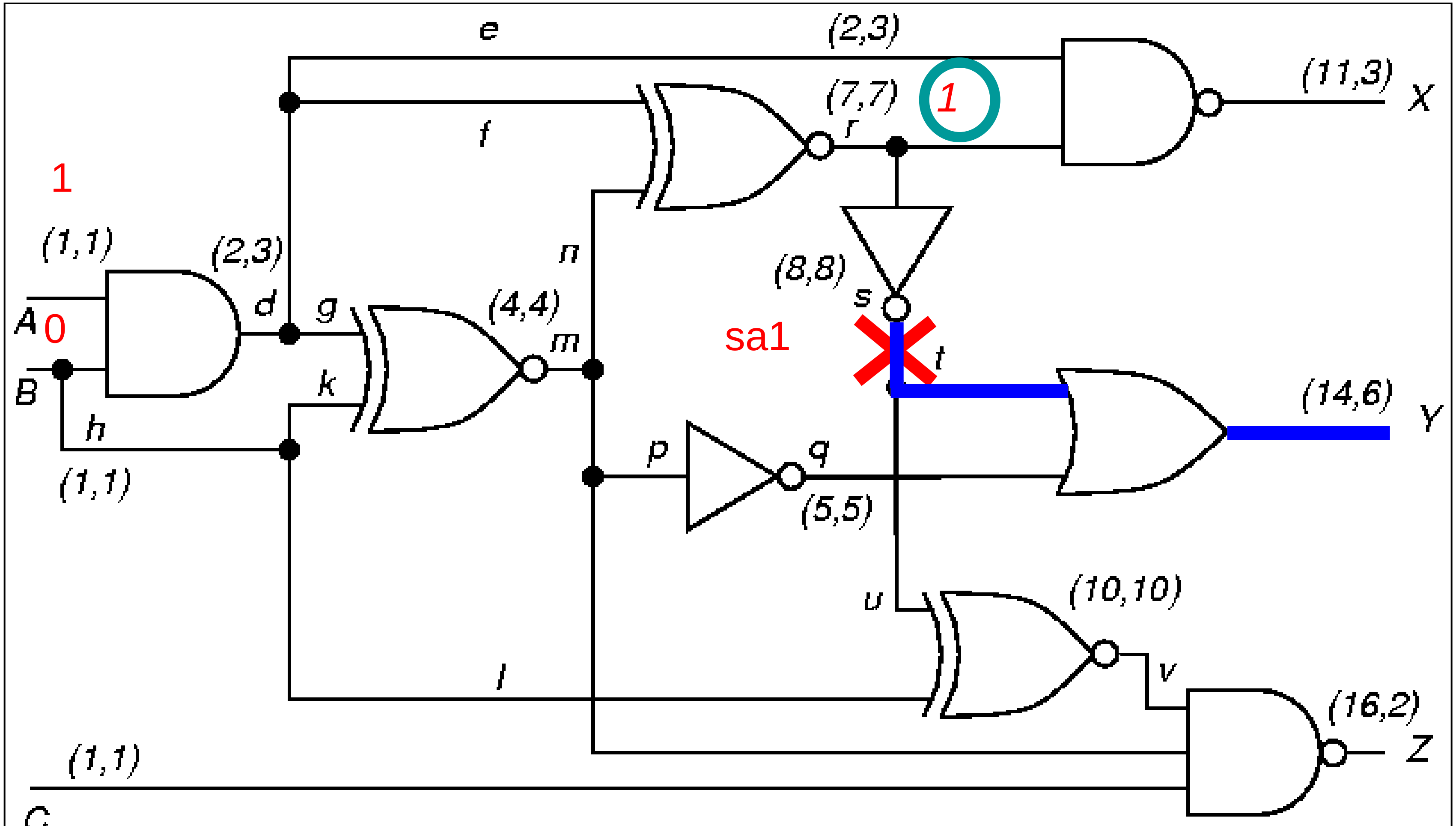
- **Backtrace from *r* again**



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

Step 15 -- s sa1

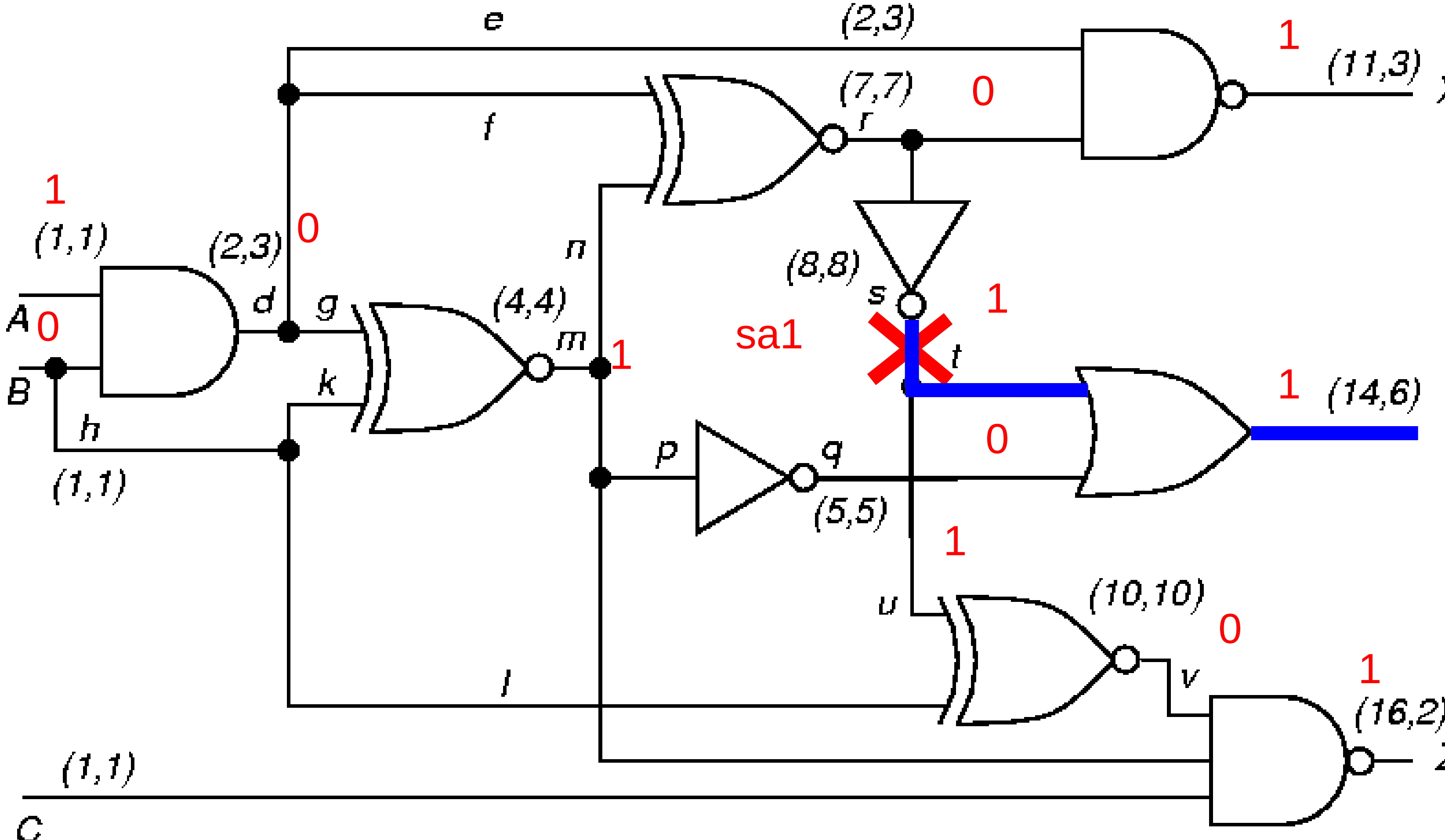
- **Set *B* = 0. Implications in stack: *A* = 1, *B* = 0**



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

Backtrack -- s sa1

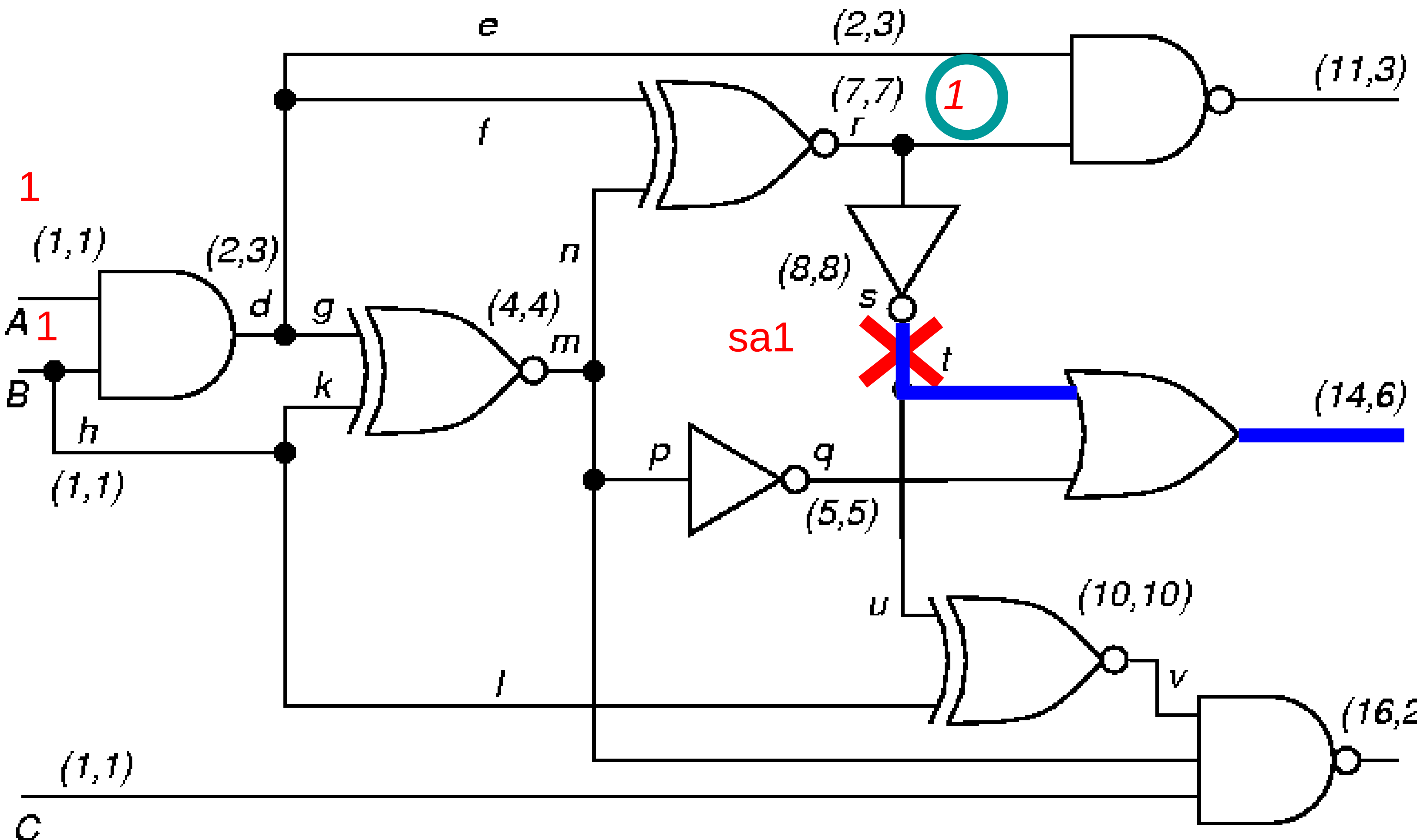
- **Forward implications: *d* = 0, *X* = 1, *m* = 1, *r* = 0. Conflict: fault not sensitized. Backtrack**



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

Step 17 -- s sa1

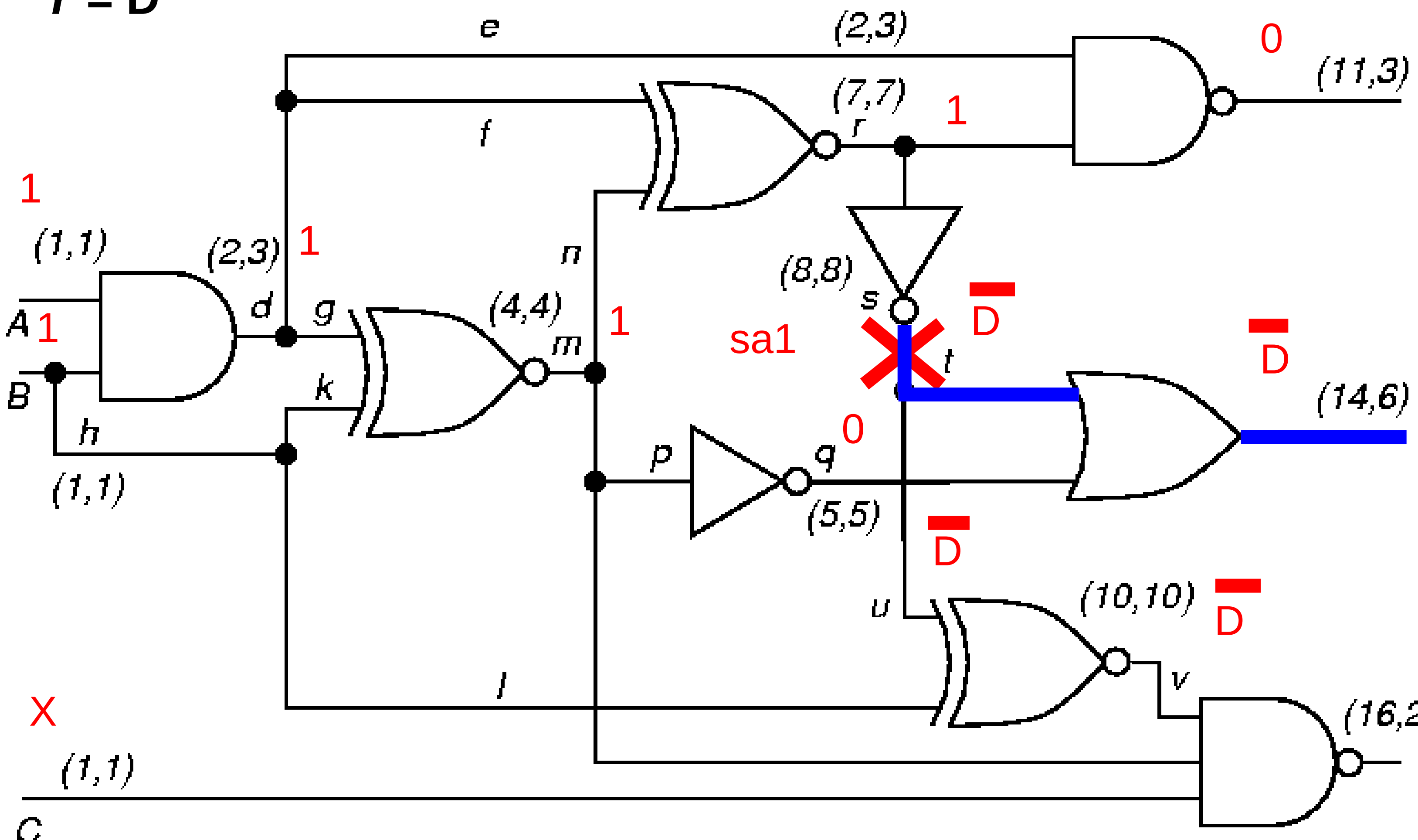
- **Set *B* = 1 (alternate assignment)**



(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* = $\overline{\textbf{D}}$, *v* = $\overline{\textbf{D}}$, *X* = 0, *Y* = $\overline{\textbf{D}}$**



(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*, $\bar{v}$);

select a gate *P* from the D-frontier;

select an unassigned input *l* 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 (*l*, $\bar{c}$);

PODEM Algorithm

while (no fault effect at POs)

    if (*xpathcheck* (D-frontier))

        (l, v_l) = *Objective* (*fault*, v_{fault});

        (pi, v_{pi}) = *Backtrace* (l, v_l);

        ***Imply* (pi, v_{pi});**

        if (*PODEM* (*fault*, v_{fault}) == SUCCESS) return (SUCCESS);

        (pi, v_{pi}) = *Backtrack* ();

        ***Imply* (pi, v_{pi});**

        if (*PODEM* (*fault*, v_{fault}) == SUCCESS) return (SUCCESS);

        ***Imply* (*pi*, “X”);**

        return (FAILURE);

    else if (implication stack exhausted)

        return (FAILURE);

    else *Backtrack* ();

return (SUCCESS);

FANout oriented test generation

FAN

(Fujiwara and Shimono, 1983)

TG Algorithms

Objective

- TG time reduction
  - Reduce number of backtracks
  - Find out the non-existence of solution as soon as possible
  - Branch and bound

FAN Algorithm

- **New concepts:**
  - **Immediate assignment of *uniquely-determined signals***
  - ***Unique sensitization***
  - **Stop Backtrace at *head lines***
  - ***Multiple Backtrace***

Thank You