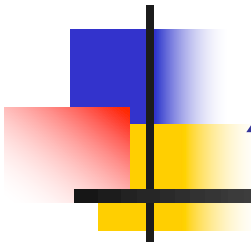
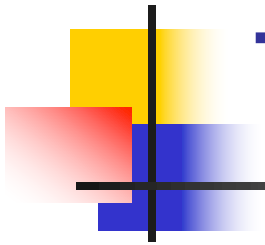


Chapter 5
ATPG

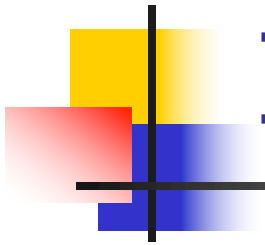
Arnaud Virazel

virazel@lirmm.fr



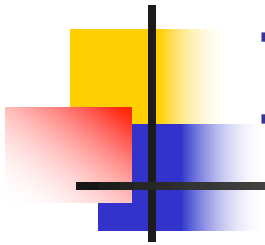
Test Pattern Generation

- Introduction
- Random Test
- Deterministic Test
  - at structural level
    - Combinatorial circuits (D-algorithm, PODEM)
    - Sequential circuits (Chapter 6)
  - at functional level
    - Memory test (Chapter 8)



Introduction

- Hypotheses
  - The test applied off-line
  - Test vectors (and goldel responses) are stored in the ATE
  - The ATE applies the test sequence and compares the test rfesponses with goldel ones



Introduction

- Main issues
  - Test generation cost (i.e. time + silicon)
  - Test quality (i.e. manufacturing defect coverage)
  - Test application cost (i.e. ATE + time)
- Evaluation
  - Test Coverage → TC

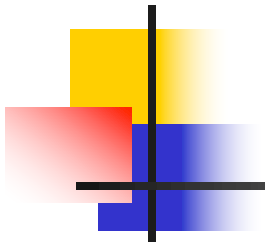
Evaluation

- Test Coverage

$$TC = \frac{\text{\# detected faults by the test sequence}}{\text{\# total of fault of the considered model}}$$

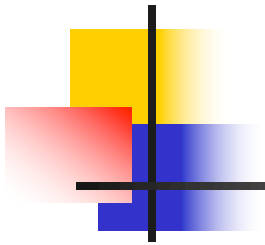
- Efficiency

$$Eff = \frac{\text{\# detected faults by the test sequence}}{\text{\# of testable faults of the considered model}}$$



Radom Test

- Test vectors generated randomly
- Low generation cost

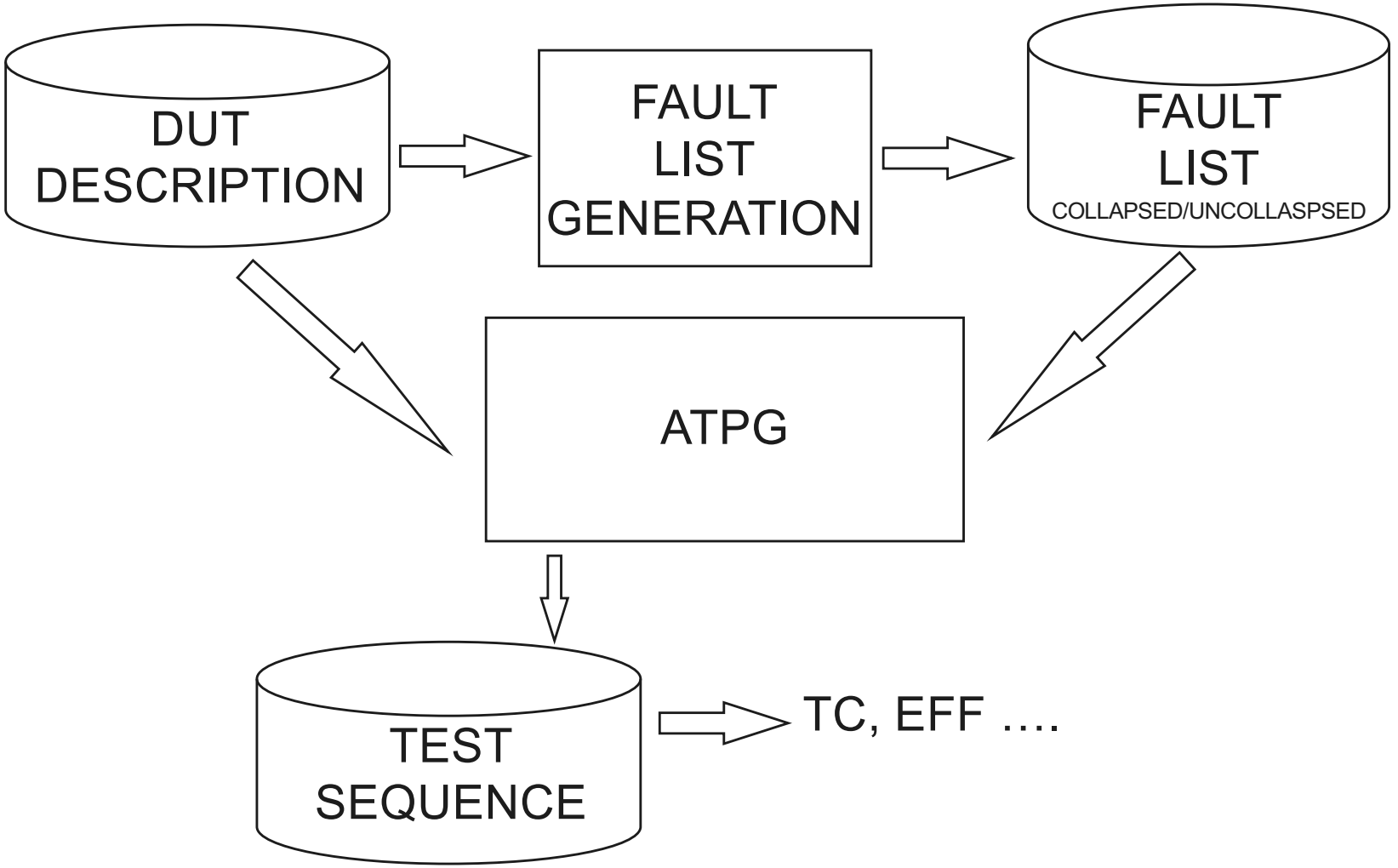


Random Test

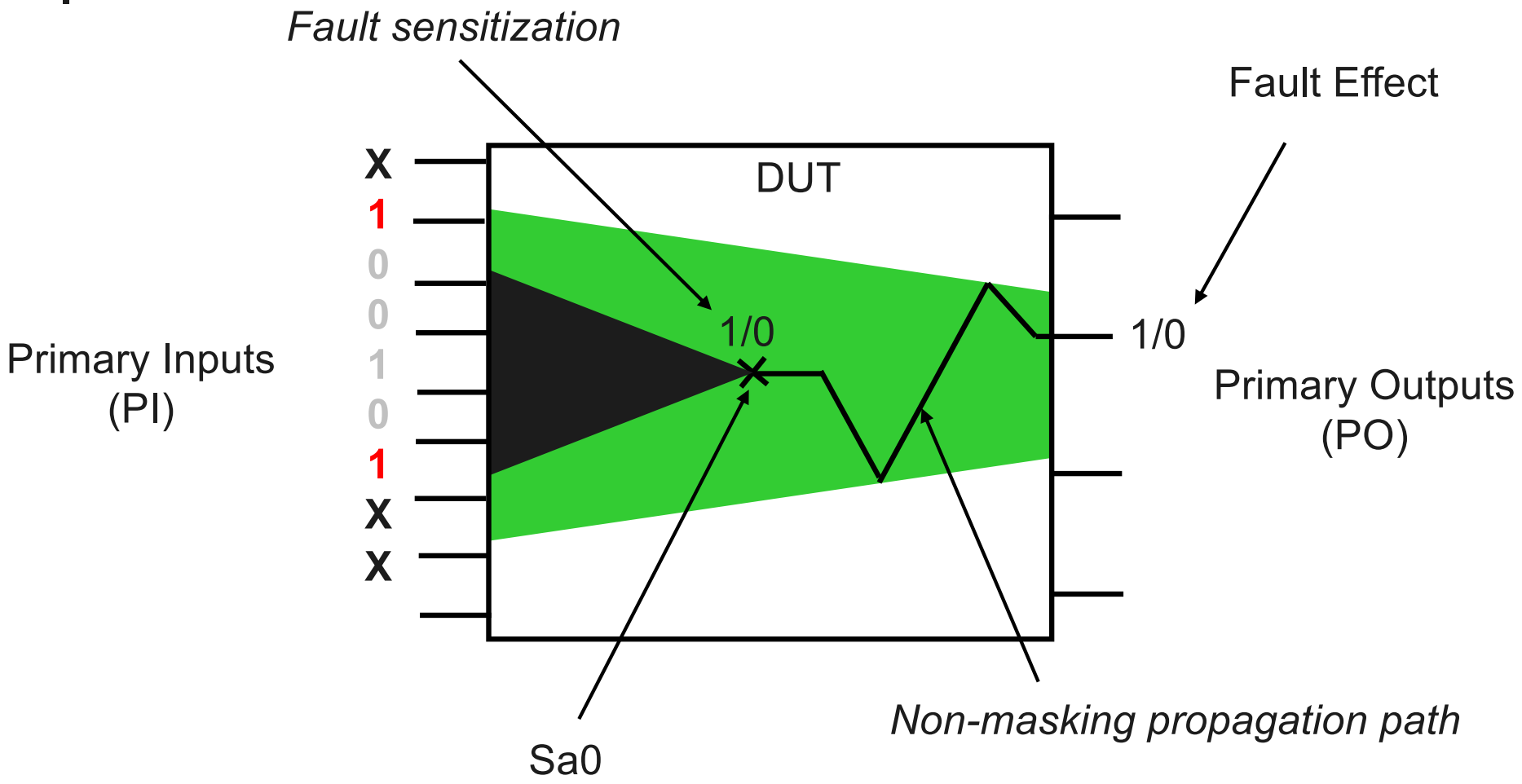
- Test quality
  - Depends on the test sequence lenght
    - It is easy to reach Tc ≅ 60% to 80%
    - Long test sequence for a reasonable Tc ≅ 95%
    - Very long test sequence for a high Tc >98%
- High application cost because long test sequences
  - High application time and memory required

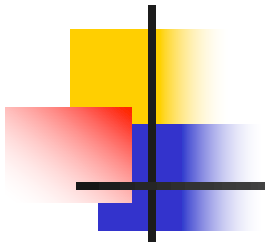
ATPG – Determinictic Test

- ATPG : **A**utomatic **T**est **P**attern **G**eneration



ATPG Process





D-algorithm – Principle

- Steps
  1. define the test of a fault f in terms of I/O of the faulty gate
  2. determine all sensitizable paths from the site of the fault to all the POs of the circuit (forward-trace phase)
  3. build the test vector on the EPs which performs all the assignments made in 1/ and 2/ (backward-trace phase)

D-algorithm – Principle

- A 5-value algebra
  - 0, 1, X, D (1/0), $\overline{D}$ (0/1)
  - D ➔ 1 golden DUT and 0 faulty DUT
  - $\overline{D}$ ➔ 0 golden DUT and 1 faulty DUT
  - $\overline{D}$ → D*
- 2 D-Cubes
  - Primitive D-cube
  - Propagation D-Cube

Primitive D-cube

- Primitive D-cube of a fault on a gate
  - Allows to propagate a fault effect, at the output of a gate using D or $\overline{D}$, by applying certain values at its inputs

e1	e2	S
1	1	D

Primitive D-cube of a Sa0 fault
at the output of an AND-gate

Propagation D-Cube

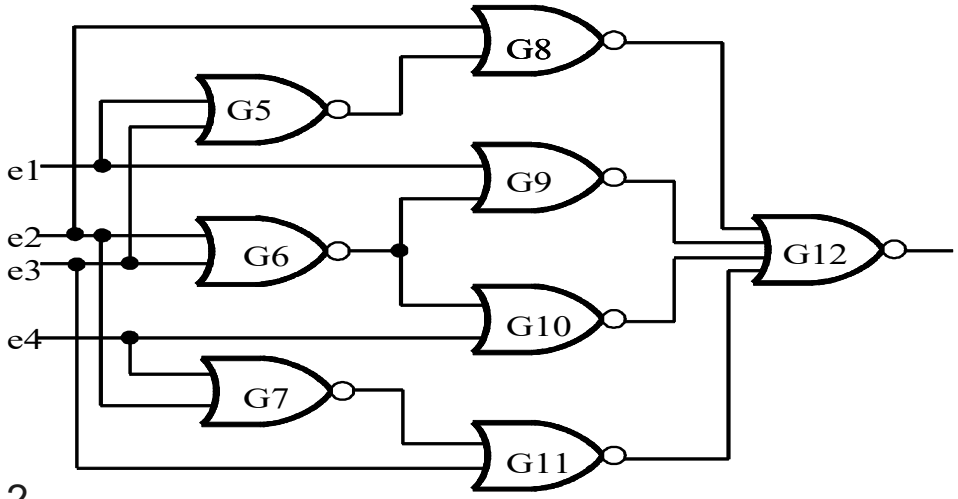
- Propagation D-cube of a gate
  - Specifies the values to apply to the side inputs, i.e. all except the one(s) carrying the fault effect
  - Then D or $\bar{D}$ value is propagated from input(s) to output

e1	e2	S	
D	1	D	cube single
1	D	D	”
D	D	D	cube multiple
$\bar{D}$	1	$\bar{D}$	cube single
1	$\bar{D}$	$\bar{D}$	”
$\bar{D}$	$\bar{D}$	$\bar{D}$	cube multiple

Propagation D-cube of a 2-input AND gate

D-algorithm – Example

- 0/ Build all the propagation D-cube of the DUT



Lines

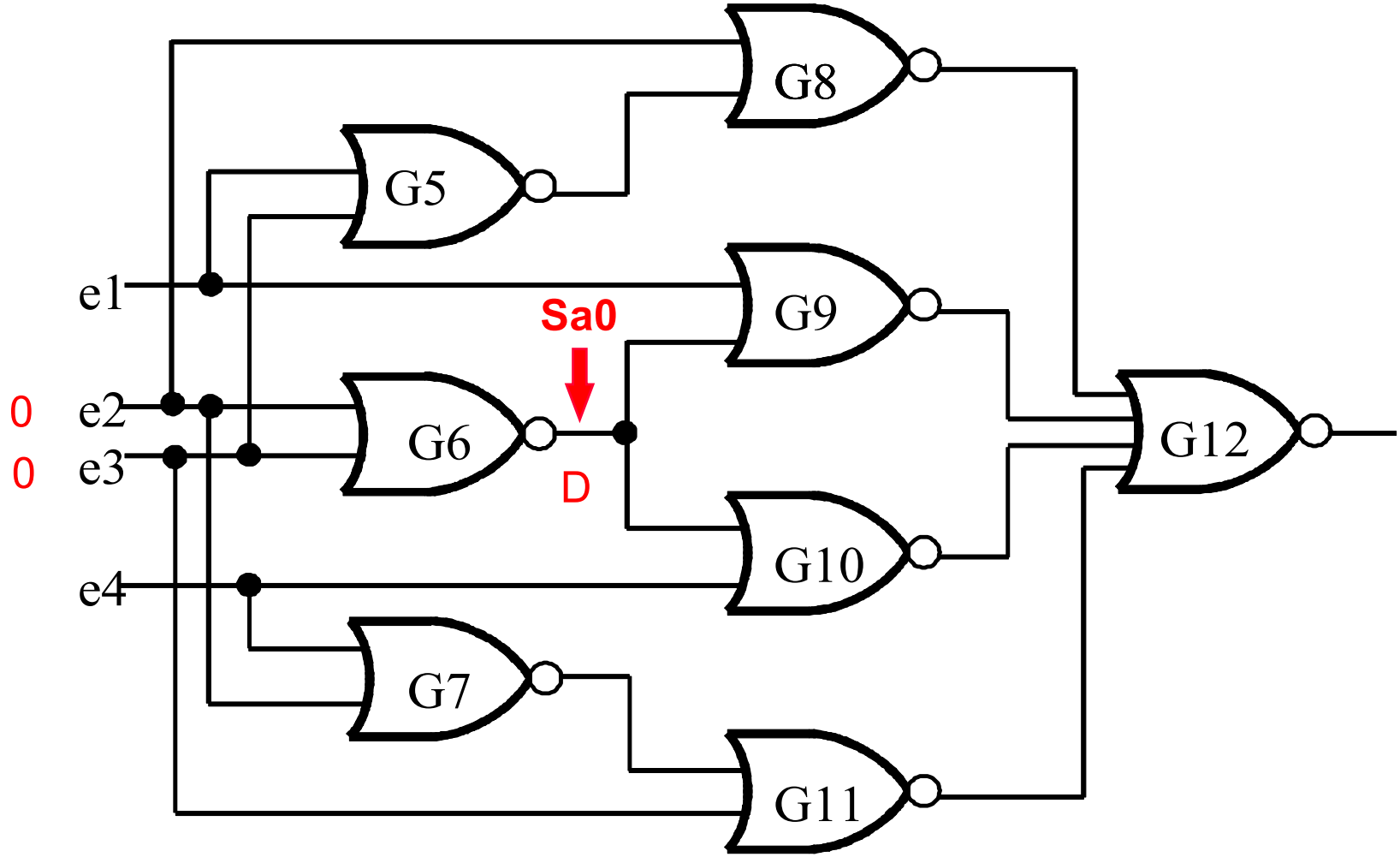
Cubes	e1	e2	e3	e4	l5	l6	l7	l8	l9	l10	l11	l12
a	0	x	D	x	D*	x	x	x	x	x	x	x
b	D	x	0	x	D*	x	x	x	x	x	x	x
c	x	0	D	x	x	D*	x	x	x	x	x	x
d	x	D	0	x	x	D*	x	x	x	x	x	x
e	x	0	x	D	x	x	D*	x	x	x	x	x
f	x	D	x	0	x	x	D*	x	x	x	x	x
g	x	0	x	x	D	x	x	D*	x	x	x	x
h	x	D	x	x	0	x	x	D*	x	x	x	x
9	**0**	**x**	**x**	**x**	**x**	**D**	**x**	**x**	**D***	**x**	**x**	**x**
.....												

D-algorithm – Example

- 1/ Select the primitive D-cube that sensitize the considered fault (i.e. Sa0 on net l6):

e1	e2	e3	e4	l5	l6	l7	l8	l9	l10	l11	l12
x	0	0	x	x	D	x	x	x	x	x	x

D-algorithm – Example



D-algorithm – Example

- 2/ Propagation of D or D* to the DUT output, i.e. l12 **forward-trace phase**

	D-cubes												Activated gate	Gate to activate	
	e1	e2	e3	e4	l5	l6	l7	l8	l9	l10	l11	l12			
C0		0	0			D							G6	G9,G10	I
C1=C0 ∩(9)	0	0	0			D			D*				G9	G12	II
C2=C0 ∩(10)		0	0	0		D				D*			G10	G12	
C3=C1 ∩(12)	0	0	0			D		0	D*	0	0	D	G12	∅	
C4=C2 ∩(12)		0	0	0		D		0	0	D*	0	D	G12	∅	
C5=C0 ∩(9) ∩(10)	0	0	0	0		D			D*	D*			G9,G10	G12	III
C6=C5 ∩(12)	0	0	0	0		D		0	D*	D*	0	D	G12	∅	

I : Initial D-cube (C0) that sensitize the Sa0 at the output of gate G6

II : 2 single paths (single D-cube) – C3 and C4 D-cubes

III : 1 multiple path (multiple D-cube) – C6 D-cube

D-algorithm – Example

- Intersection rules

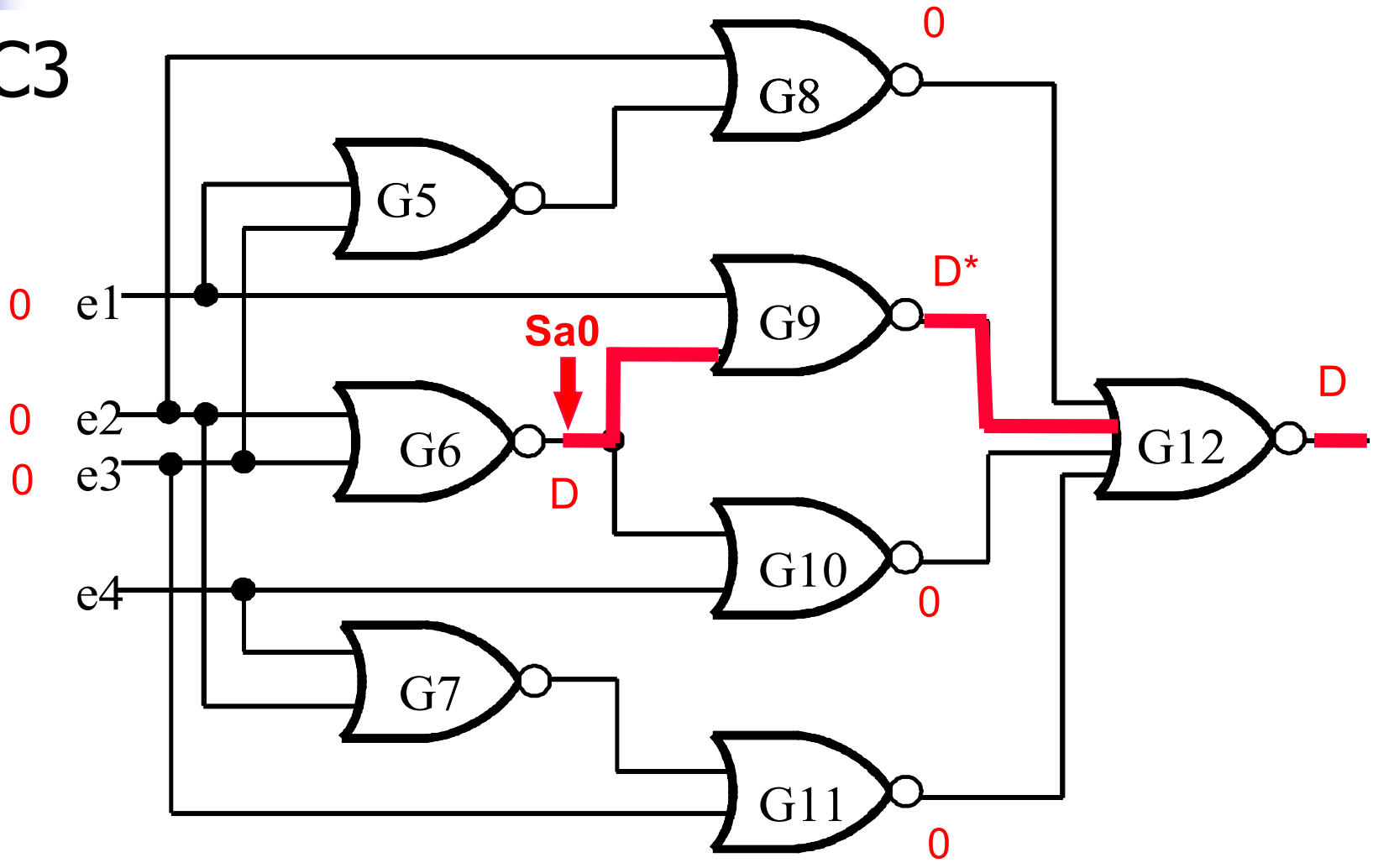
∩	0	1	x	D	D*
0	0	-	0	-	-
1	-	1	1	-	-
x	0	1	x	D	D*
D	-	-	D	D	-
D*	-	-	D*	-	D*

conflict

	e1	e2	e3	e4	l5	l6	l7	l8	l9	l10	l11	l12
C0	x	0	0	x	x	D	x	x	x	x	x	x
(9)	0	x	x	x	x	D	x	x	D*	x	x	x
∩ =	**0**	**0**	**0**	**x**	**x**	**D**	**x**	**x**	**D***	**x**	**x**	**x**

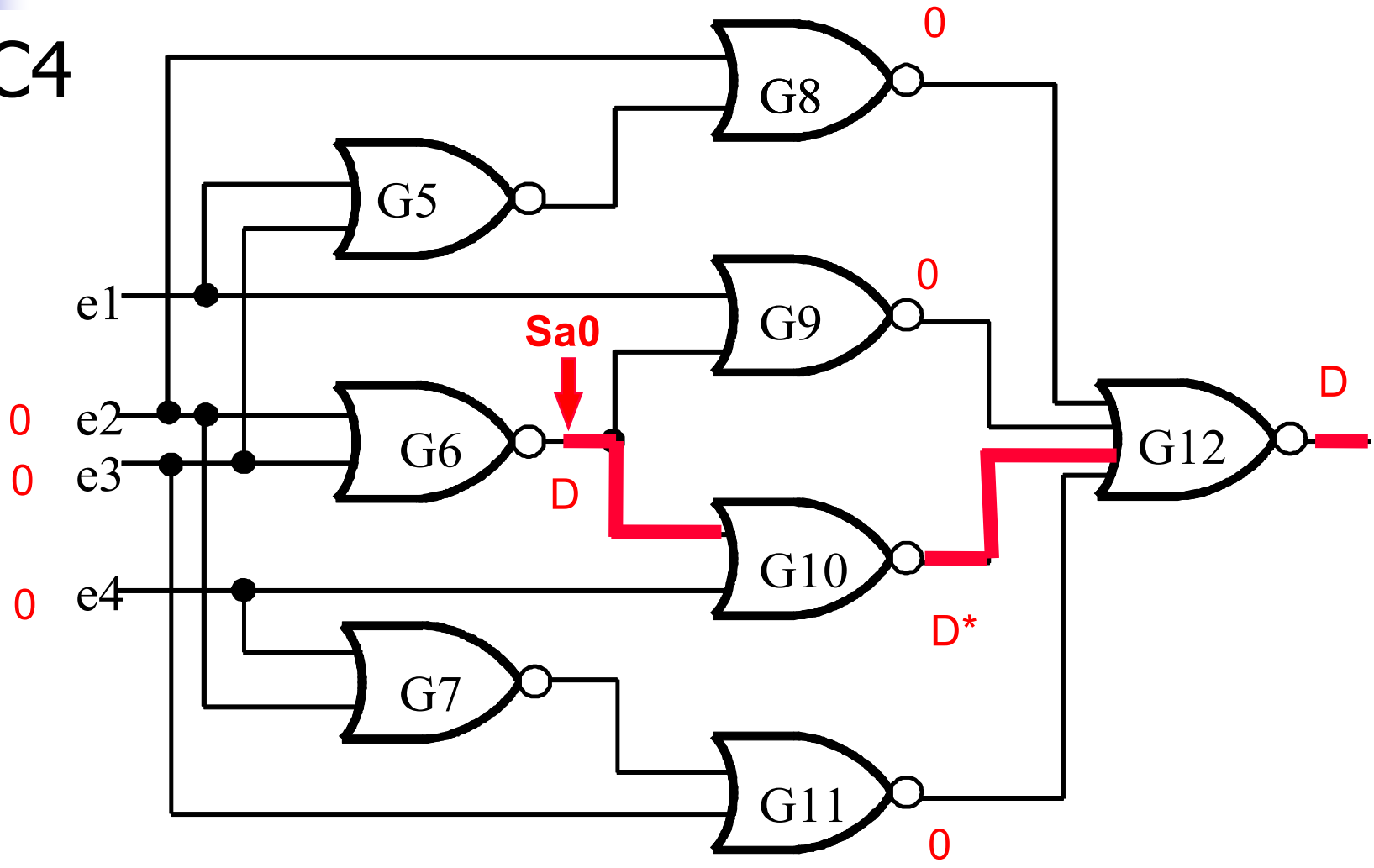
D-algorithm – Example

- C3



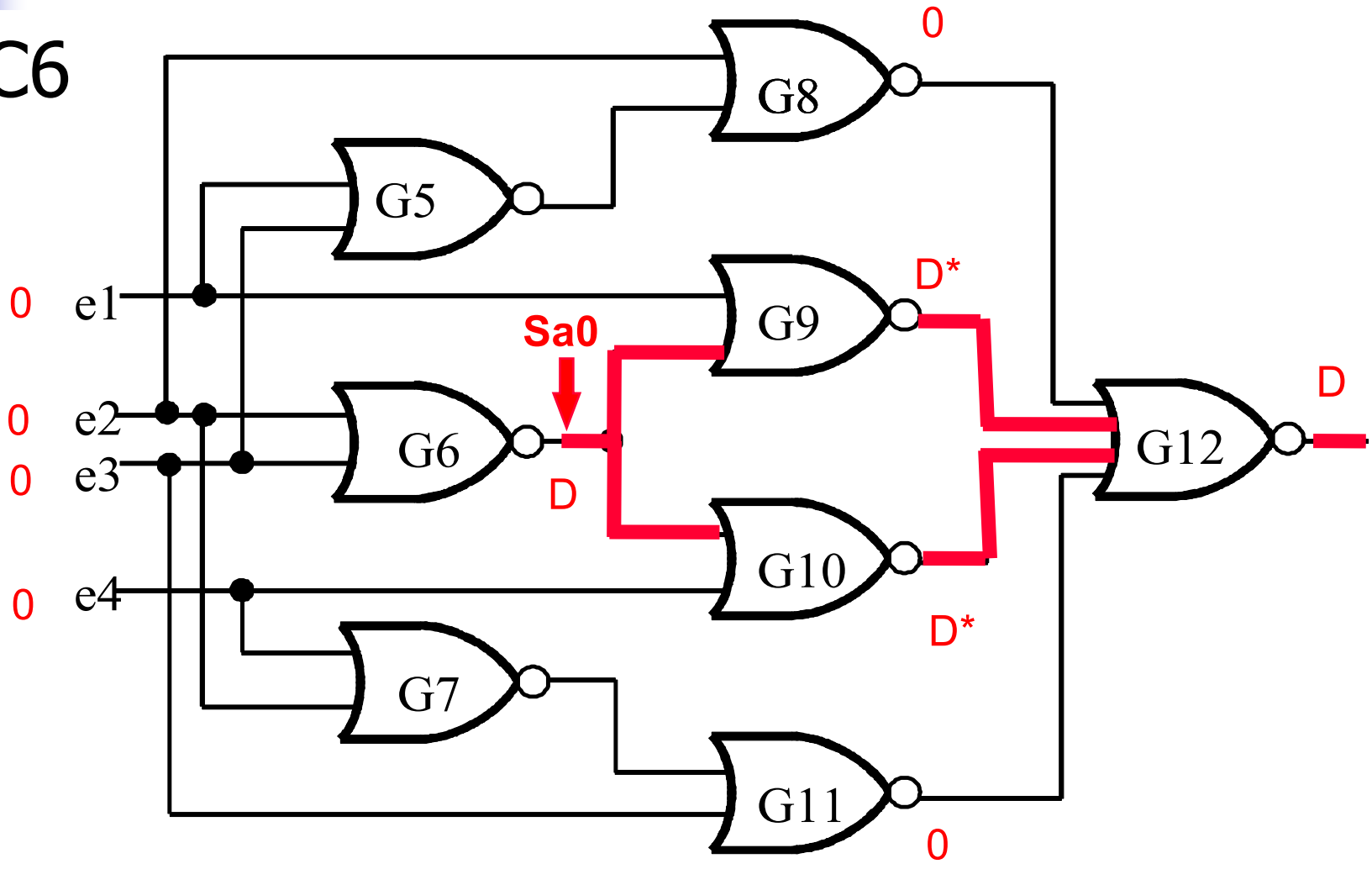
D-algorithm – Example

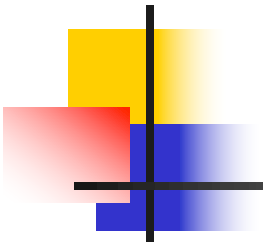
- C4



D-algorithm – Example

- C6





D-algorithm – Example

- 3/ build the test vector on the EPs which performs all the assignments made during step 1/ and 2/
  backward-trace phase

D-algorithm – Example

3/ Justification phase of D-cube C3

	D-cubes											
	e1	e2	e3	e4	l5	l6	l7	l8	l9	l10	l11	l12
C3	0	0	0	x	x	D	x	0	D*	0	0	D
P11(a) NOR G11	x	x	x	x	x	x	1	x	x	x	0	x
P11(b) NOR G11	x	x	1	x	x	x	x	x	x	x	0	x
C3 ∩ P11(a)	0	0	0			D	1	0	D*	0	0	D
P10(a) NOR G10				1		x				0		
P10(b) NOR G10				x		1				0		
C3 ∩ P11(a) ∩ P10(a)	0	0	0	1		D	1	0	D*	0	0	D
P8(a) NOR G8		1			x			0				
P8(b) NOR G8		x			1			0				
C3 ∩ P11(a) ∩ P10(a) ∩ P8(b)	0	0	0	1	1	D	1	0	D*	0	0	D
P7 NOR G7		0		0			1					

Value to justify

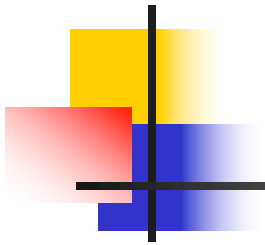
Incompatibility

Incompatibility on PI e4
Unable to test Sa0 on l6 using the single propagation path corresponding to D-cube C3 (G9, G12)

D-algorithm – Example

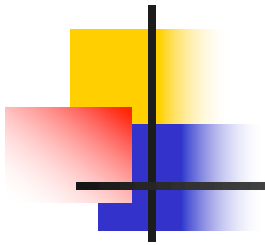
3/ Justification phase of D-cube C6

Cube	D-intersection											
	1	2	3	4	5	6	7	8	9	10	11	12
C6	0	0	0	0		D		0	D̄	D̄	[0]	D
P11(a)		1					x				0	
P11(b)		x					1				0	
C6∩P11(b)	0	0	0	0		D	1	[0]	D̄	D̄	0	D
P8(a)		1			x			0				
P8(b)		x			1			0				
C6∩P11(b)∩P8(b)	0	0	0	0	1	D	[1]	0	D̄	D̄	0	D
P7		0		0			1					
C6∩P11(b)∩P8(b)∩P7	0	0	0	0	[1]	D	1	0	D̄	D̄	0	D
P5	0		0		1							



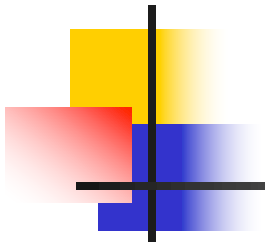
D-algorithm – Issues

- Excessive execution time
  number of paths + exhaustive process + redundant faults
- Improvements
  - 9V- algorithm
    same strategy but simultaneous sensitization of all propagation paths
  - A new strategy with PODEM
    Path-Oriented Decision Making



PODEM – Principle

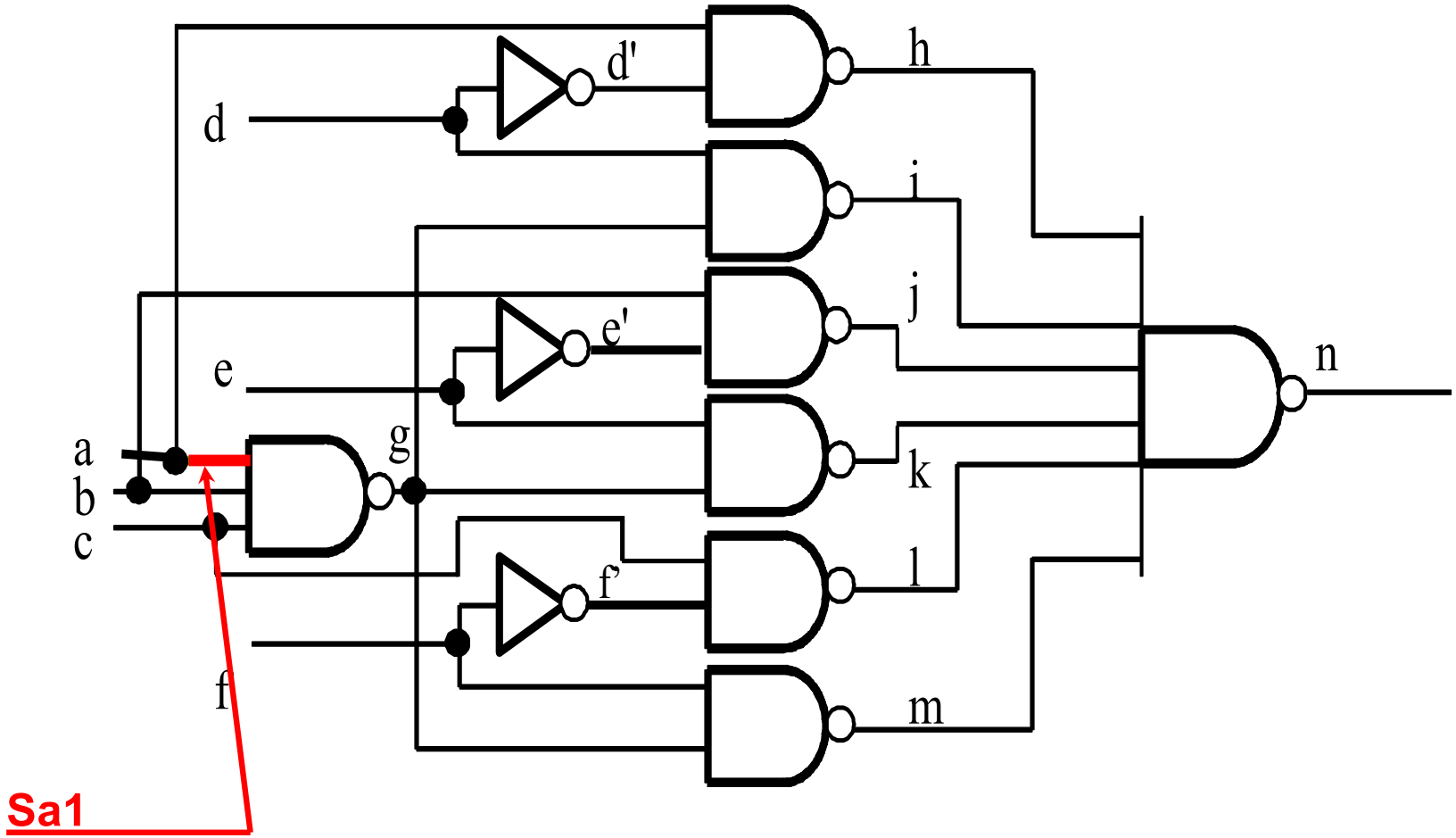
- The test of combinatorial circuits is seen as a traveling problem in a tree
- Algorithm in which all input combinations are implicitly examined as a test vector for a given fault
- The circuit structure is used to guide successive trials of input combinations



PODEM – Algorithm

1. Determine the objective to be achieved
   - propagation of a D value to POs
2. «backward» of the objective to PIs
   - If justification of the inputs of a gate whose output is at a priority value (i.e. for example a 1 for the OR gate), select the most easily controllable input at the priority value → among several solutions, select the one that is most likely to be satisfied
   - If justification of the inputs of a gate whose output is at a non-priority value (i.e. for example a 0 for the OR gate), select the most difficult input to control at the non-priority value → among several problems to be solved select of the one to be the most difficult (in order to avoid global failure after solving easier problems)
3. Compute the implications of all assignments made in /2
4. If a test vector is found, exit
5. Else
   - 5.1. if the objective is reached then modification an back to /1
   - 5.2. Else
     - if test still possible then return to /2 for a new assignment
     - Else return to /1 for modification of the objective

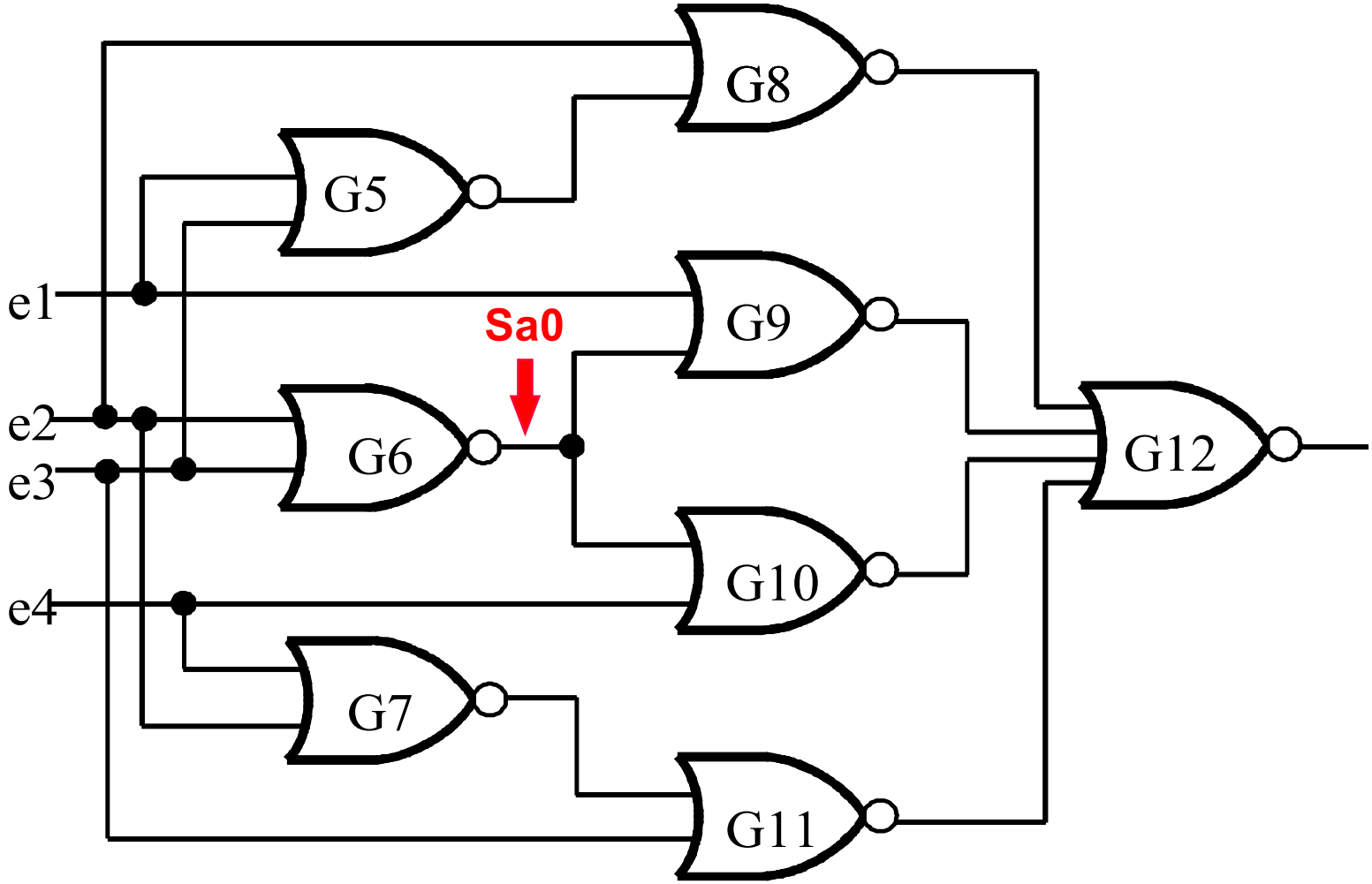
PODEM – Example

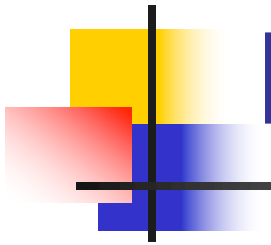


PODEM – Algorithm

Objective	PIs	Implications	D-border	
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	
j=1	e=1	e'=0 k=D*	m, n	
l=1	f=1	f'=0 m=D* **n=D**		Test generation success

PODEM – Exercise





FAN

- It is an improvement of PODEM on a certain number of heuristics relating to divergences (i.e. fanout)

- Improvements
  - The justification phase is stopped on the tops of logic cones of the DUT (i.e. before any fanout) because their justification will always be possible → CPU time saving
  - Multiple propagation → all branches of a fanout are exploited