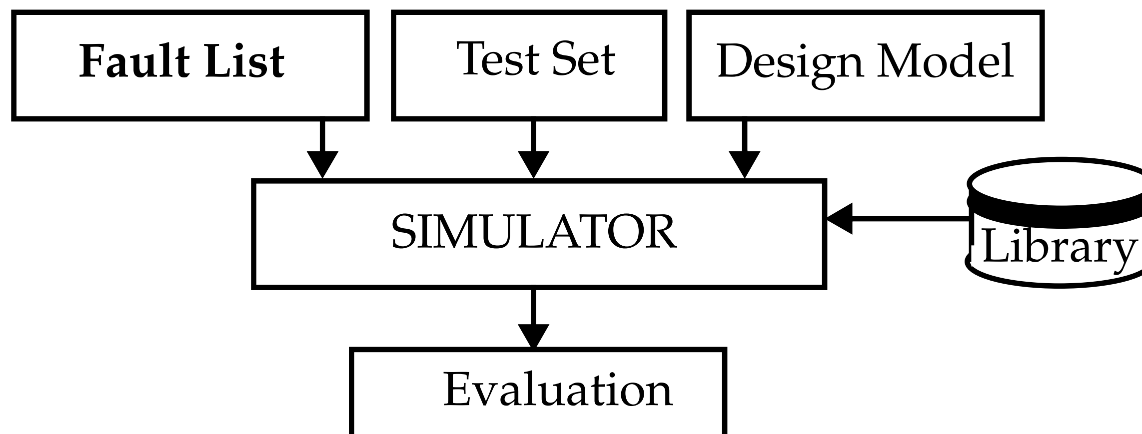
Algorithms for Fault Simulation

Purposes of fault simulation during design cycle:

- Guiding the TPG process.
- Measuring the effectiveness of the test patterns.
- Generating fault dictionaries.

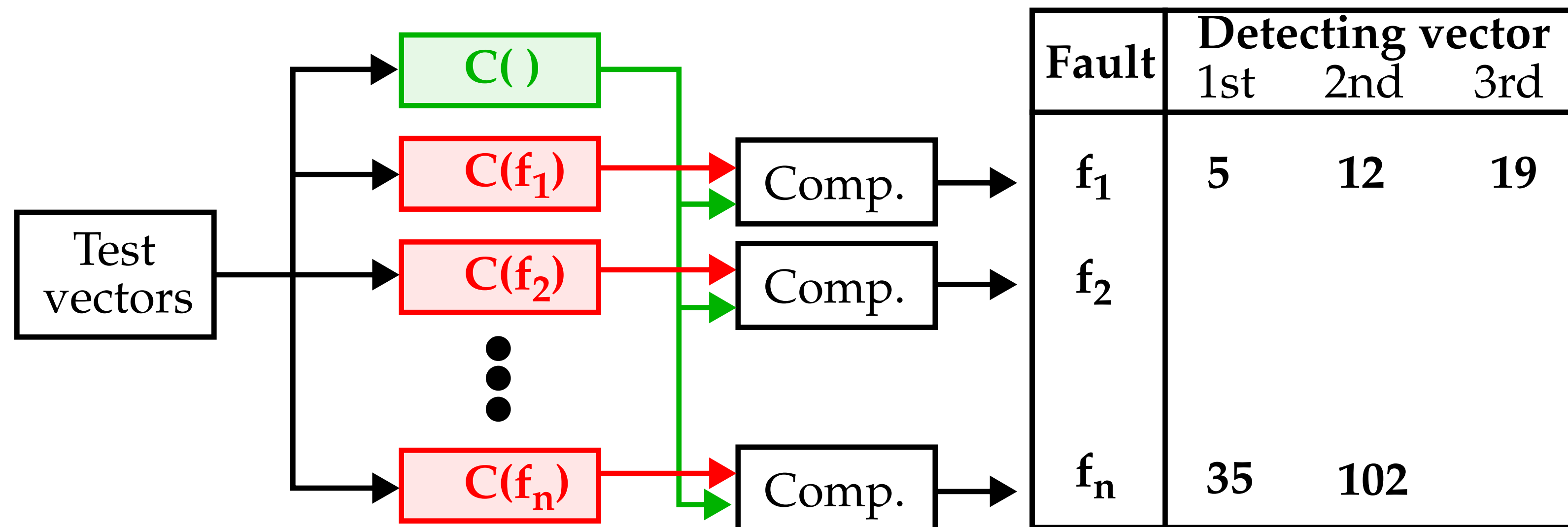
Fault simulator needs in addition to the circuit model, stimuli and expected responses (that are needed for true-value simulation):

- Fault model
- Fault list



Algorithms for Fault Simulation

The fault simulator must classify the given target faults as *detected* or *undetected* by the given stimuli.



$C(f_1)$...$C(f_n)$ are copies of the defect-free circuit C( ) with fault f_x permanently inserted.

Here, each time the fault is detected, the simulator records the vector number (and possibly the output(s) in error).

Although useful for **fault diagnosis**, this is compute expensive.

Fault dropping causes simulation of $C(f_n)$ to stop after vector 35.

Serial Fault Simulation

If fault dropping is not employed, the effort of simulating n faults is equivalent to either:

- Simulating a circuit n times larger or
- Repeating the original true-value simulation n times.

Serial Fault Simulation

True-value simulation is performed across all vectors and outputs saved.

Faulty circuits are simulated one-by-one by modifying circuit and running true-value simulator.

Simulation of faulty circuit stops as soon as fault is detected.

Adv:

Any type of fault can be simulated, e.g., stuck-at, stuck-open, bridges, delay and analog faults.

For n faults, CPU time can be almost n times that of a true-value simulator.

Fault dropping significantly improves on this.

Parallel Fault Simulation

Most effective when:

- Circuit consists of only logic gates.
- Stuck-at faults are modeled.
- Signals assume only binary, 0 or 1, values.
- All gates have the same delay (zero or unit).

Under these conditions, circuits $C(f_n)$ are almost identical.

Here, the bit-parallelism of logical operations in a computer can be useful.

For a 32-bit word, 1 fault-free and 31 faulty circuits can be simultaneously simulated.

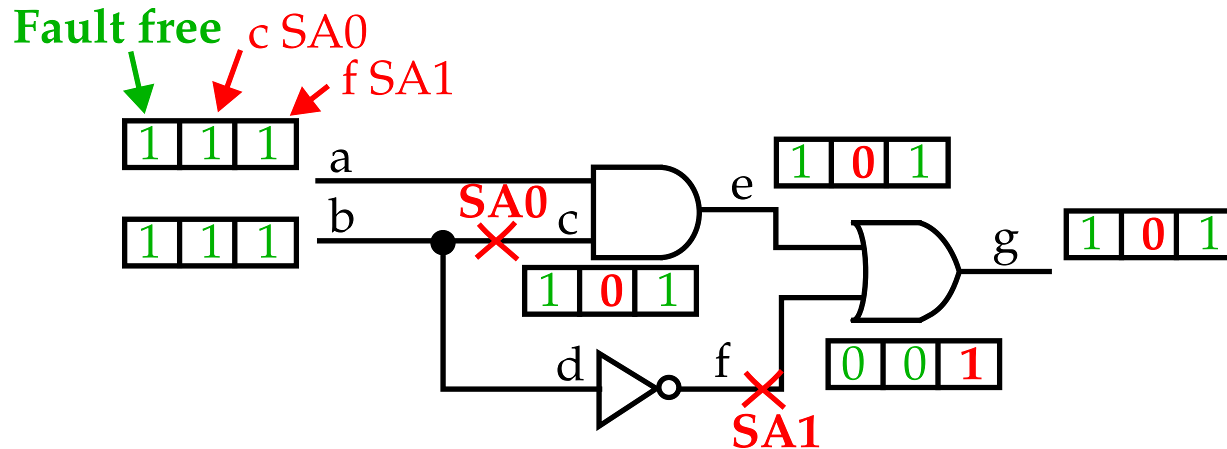
This yields a speed up of w - 1, with w equal to the word size.

If fault dropping is employed, simulation stops when all w - 1 faults are detected.

Therefore, serial fault simulation has more to gain by fault dropping.

Parallel Fault Simulation

Parallel fault simulation of two faults, *c* SA0 and *f* SA1:



Parallel fault simulation cannot accurately model rise and fall delays.

The signal values in all circuits are processed simultaneously.

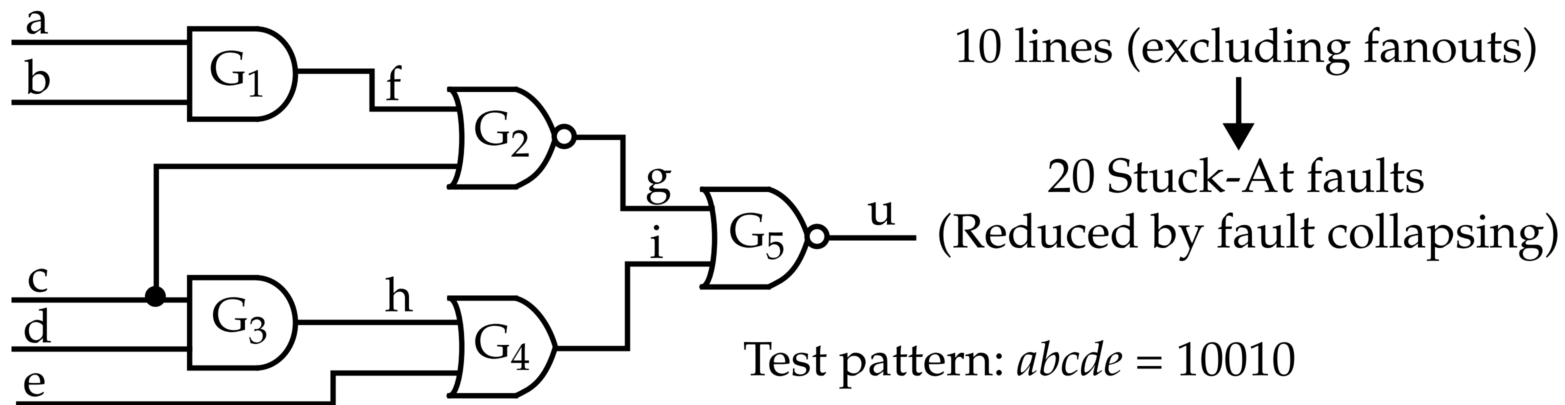
Zero-delay or unit-delay are used.

Compiled-code or event-driven versions are possible.

Multi-valued logic is possible, e.g., (0, 1, X and Z), by encoding state in more than 1 bit.

A true-value logic simulator can be used as a parallel fault simulator by inserting gates to model faults -- see text.

Parallel Fault Simulation



Reduction possible: *f* SA1 **eq** *g* SA0, *f* SA1 **dom** *b* SA1 and *u* SA0 **dom** *g* SA0.

	ff	a/0	b/1	c/1	d/0	e/1	f/0	f/1	g/0	g/1	h/0	h/1	i/0	i/1	u/0	u/1
a=1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
b=0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
c=0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
d=1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
f=ab	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
g=$\overline{f+c}$	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1
h=cd	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0
e=0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
i=e+h	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0
u=g+i	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1

Deductive Fault Simulation

Circuit model assumptions are the same as those given for the parallel fault simulator, compiled-code and event-driven versions possible.

Only the fault free circuit, C( ), is simulated.

Faulty circuit values are deduced from the fault-free values.

It processes all faults in a **single pass** of true-value simulation, i.e., it very fast!

Note, however, that major modifications are required (and slow downs) to handle variable rise/fall delays, multiple signal states, etc.

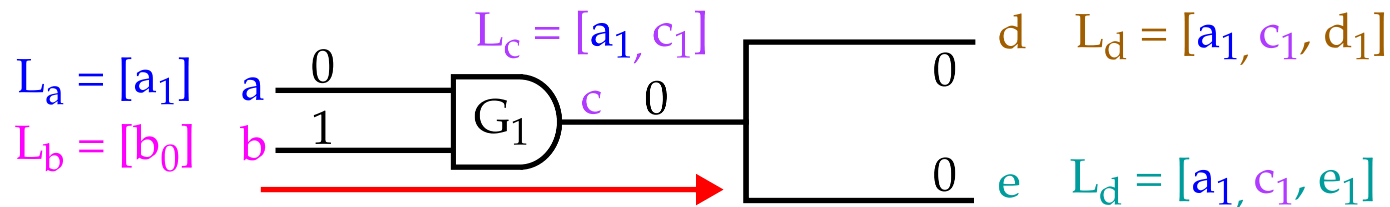
A vector is simulated in true-value mode.

A deductive procedure is then performed on all lines in level-order from inputs to outputs.

Fault lists are generated for each signal using the fault lists on the inputs to the gate generating that signal.

Deductive Fault Simulation

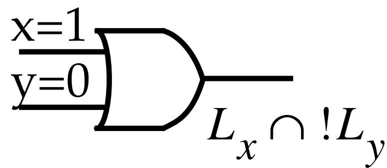
The fault list of a signal contains the *names* of all faults in the circuit that can change the state of that line.

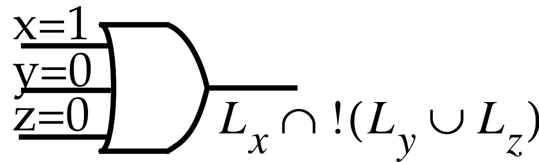


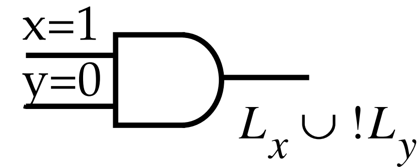
Gate type	Inputs		Output	Output fault list L_c
	a	***b***	***c***	
AND	0	0	0	[L_a intersection L_b] union c_1
	0	1	0	[L_a intersection $\overline{L}_b$] union c_1
	1	0	0	[$\overline{L}_a$ intersection L_b] union c_1
	1	1	1	[L_a union L_b] union c_0
OR	0	0	0	[L_a union L_b] union c_1
	0	1	1	[$\overline{L}_a$ intersection L_b] union c_0
	1	0	1	[L_a intersection $\overline{L}_b$] union c_0
	1	1	1	[L_a intersection L_b] union c_0
NOT	0	-	1	L_a union c_0
	1	-	0	L_a union c_1

Deductive Fault Simulation

For example, if both inputs to a 2-input AND are 0, in order for a fault to propagate through, it must be in the lists of both inputs.

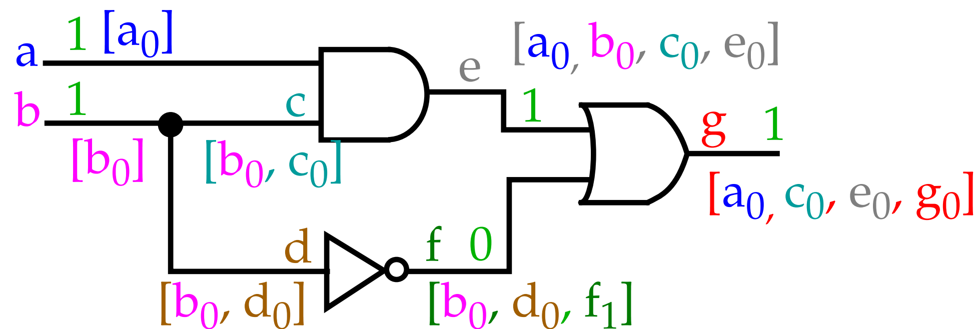






Fault-list, L_x, intersection $!L_y$ is equivalent to $L_x \cap !L_y = L_x - (L_x \cap L_y)$

True-value simulation is run first.

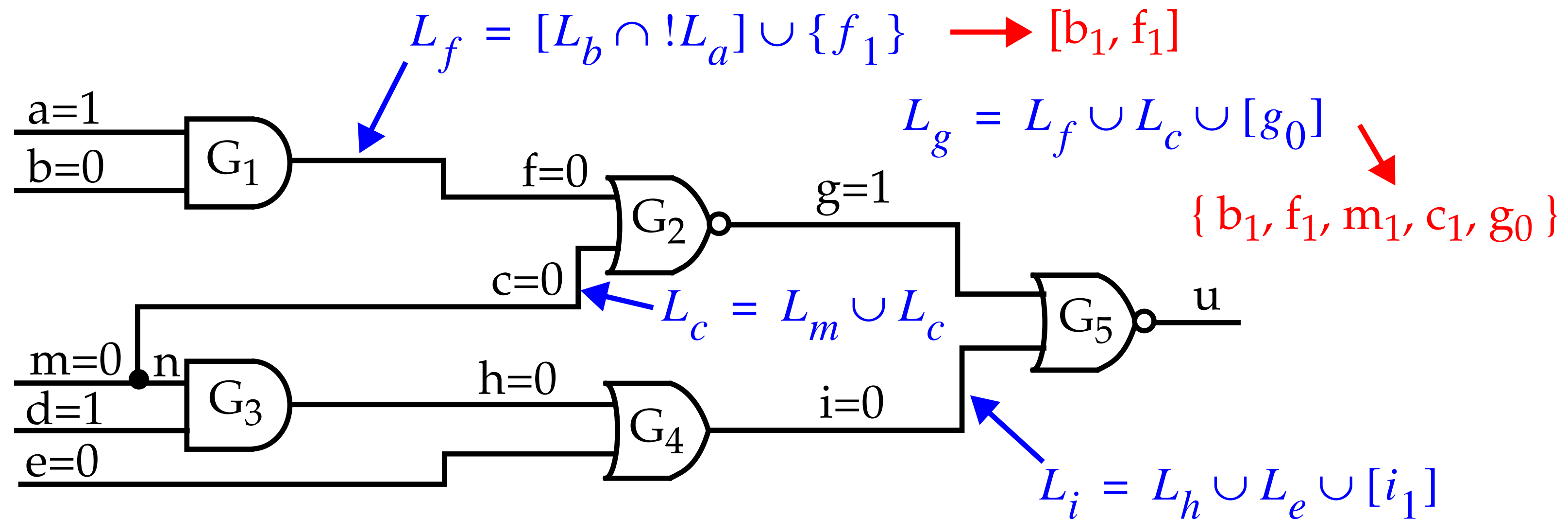


Fault list for *e* is composed from the union of the input lists for *a* and *c*, since the input is *ab* = (11).

Fault list for *g* is given by the intersection of *e*'s list and !*f*'s list.

Deductive Fault Simulation

Another example:



With $ab = 10$, only L_b is sensitizable to f (faults on a are masked).

The faults given by $L_f = [L_b \cap !L_a] \cup [f_1]$ are the faults in L_b that are not in L_a union $[f_1]$. Therefore, $L_f = [b_1, f_1]$.

Had $b = 1$, L_a would have been sensitized to f, e.g.,

$$L_f = L_a \cup L_b \cup [f_1]$$

Concurrent Fault Simulation

It extends the event-driven simulation method to simulation of faults.

It can handle various types of circuit models, faults, signal states and timing models.

Details of the simulator model:

- Events

  Good events: Occur in the fault-free circuit, C( ), and have three attributes, signal name, type of transition (0-to-1) and time of change.

  Fault-events: Occur on same lines in faulty circuits, $C(f_1)$...$C(f_n)$, but ONLY if transition is different from C( ) transition. Three attributes + fault site and type.

- Circuit

  Modeled in the same way as for true-value simulation.

  Each *good-gate* has a fault list of *bad-gates* associated with it.

  Bad-gates are not faulty but rather have an I/O that is affected by some fault.

Concurrent Fault Simulation

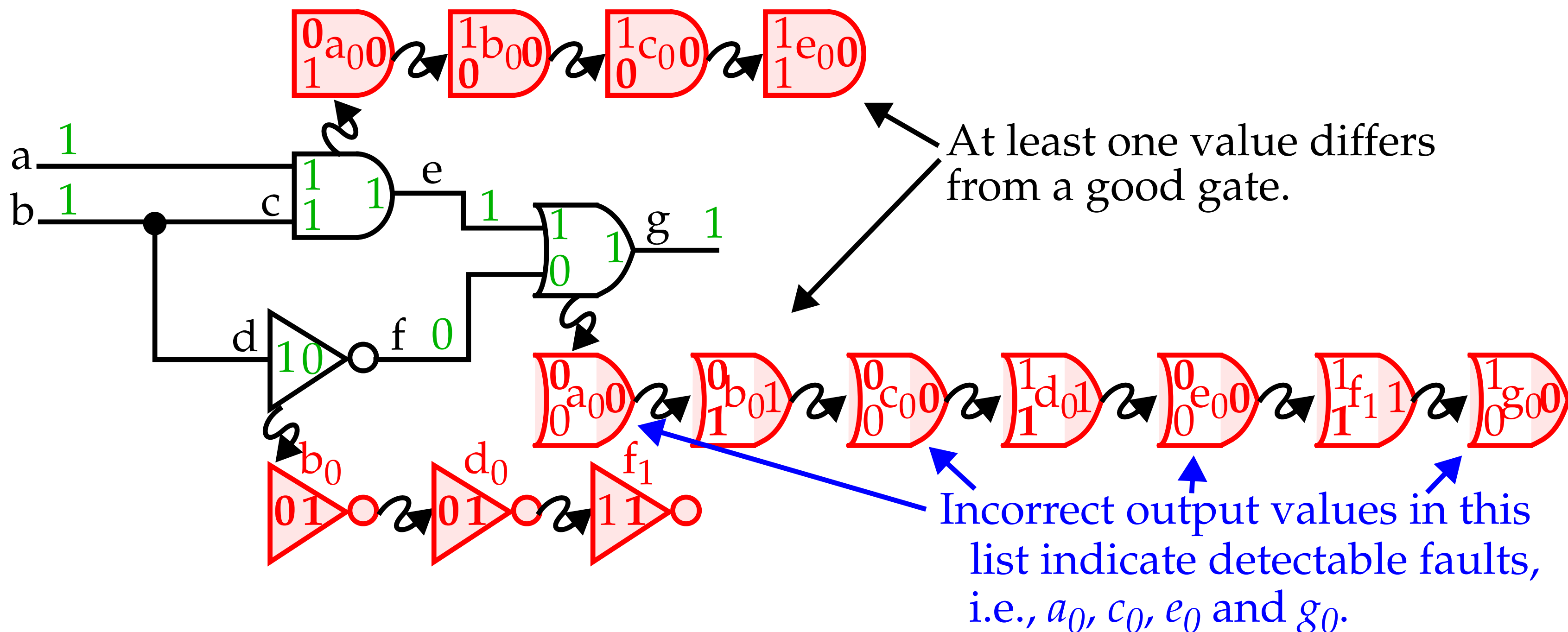
- Faults

  Whenever the signal values of a *good-gate* make a fault active, a *bad-gate* is inserted into the fault list on that *good-gate*.

Event-driven simulation is carried out.

Good-events and fault-events make good-gates active for evaluation.

Good-events also make bad-gates active for evaluation.

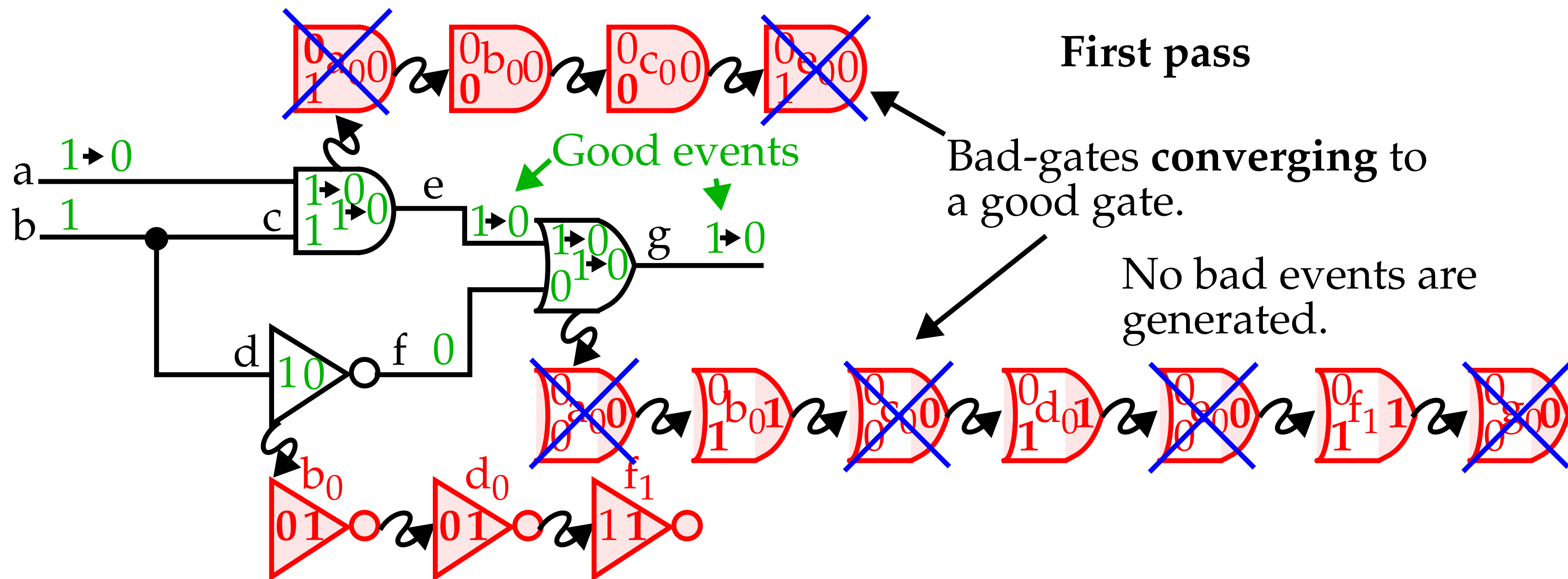


Concurrent Fault Simulation

This agrees with the deductive fault simulator results.

Note that deductive list is smaller -- fault lists include faults that affect a **signal line**.

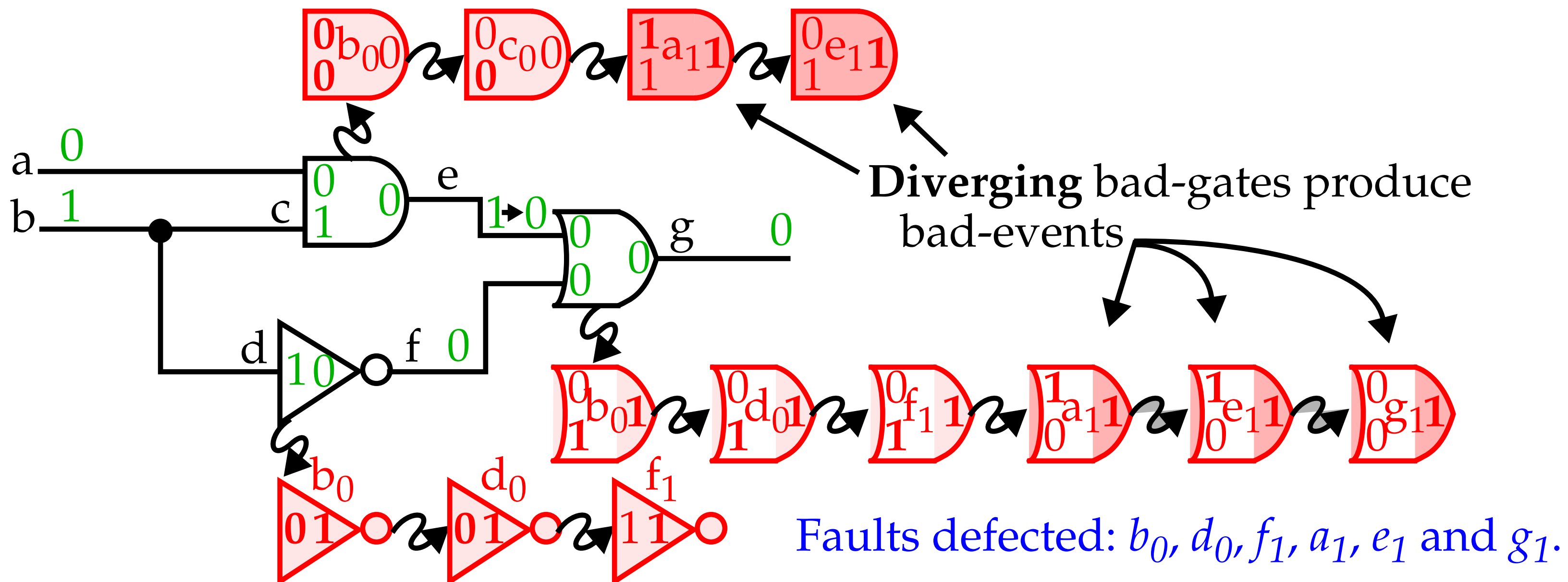
In contrast, concurrent list contains faults that affect the **gate** and includes those that affect its inputs (adv in memory, RTL and behavioral models).



Note that we did not ***drop*** a_0, c_0, e_0 and g_0 after the first simulation for illustration only.

Concurrent Fault Simulation

Second pass processes the activation of fault a_1 under the good-event transition



The most significant advantages of this algorithm are:

- Efficiency -- redundant computation is eliminated.
- Modeling flexibility -- anything that can be simulated.

MARS, CREATOR, MOZART, MOTIS and FMOSSIM are examples.

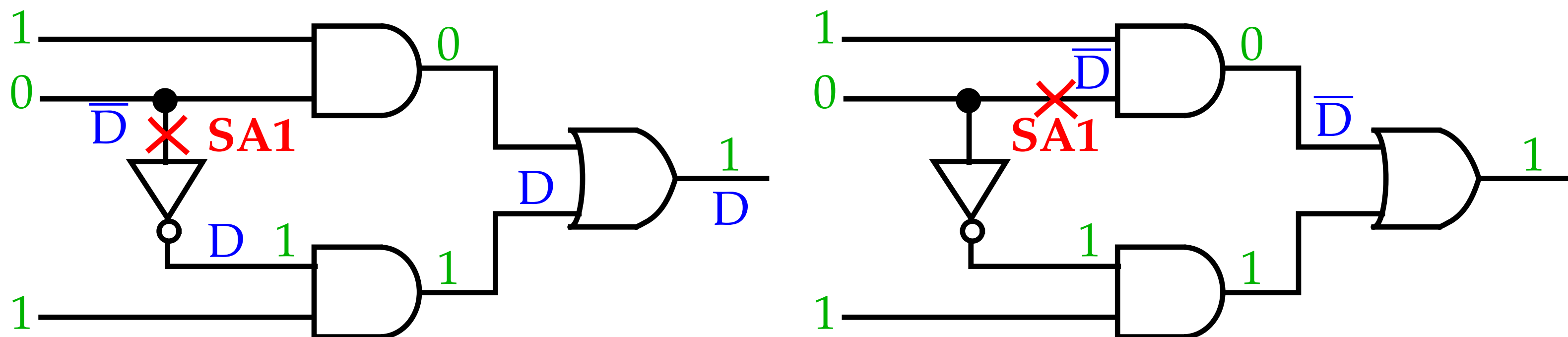
Roth's TEST-DETECT Algorithm

The circuit is simulated for a vector in true-value mode to determine node states (zero delay model is assumed).

Faults are then simulated one at a time to determine which are detected by this vector.

For *two-value* simulation, the signal state given as (*fault-free*, *faulty*) can take 4 possible assignments.

$0 = (0, 0)$, $1 = (1, 1)$, $D = (1, 0)$ and $\overline{D} = (0, 1)$



D-calculus is used to represent both fault-free and faulty values.

Starting at the fault site, if fault is activated, a *D* or $\overline{D}$ is placed there.

The symbol is propagated, if it reaches an output, fault is detectable.

Differential Fault Simulation

Cheng and Yu made 2 improvements on TEST-DETECT.

- Eliminated the use of *D*-calculus.
- Eliminated the explicit restoration to true-value before processing the next fault.

The algorithm, which starts with a vector set and a fault list:

- Simulate a vector in *true-value* mode and store the PO values.
- Activate a fault by creating a transition to the faulty value, e.g., if *true-value* is 0 and it is a SA1, generate a 0 -> 1 transition.
- Simulate the circuit and check for a difference at POs -- drop the fault if detected.
- For next fault, **restore** to true value by placing a restoring transition at previous site. Place a second transition at new fault site and simulate.
- Repeat with the next vector once all faults have been analyzed.

PROOFS: a popular, parallel implementation of this *differential fault* simulation algorithm.

Both this and TEST-DETECT can handle synchronous sequential circuits.