ATPG

Automatic Test Pattern Generation has several purposes:

- It can generate test patterns (obviously)
- It can find redundant circuit logic.
- It can prove one implementation matches another.

Why is ATPG necessary?

Complete functional test is impractical.

Designer generated functional patterns typically provide only 70-75% SA coverage.

ATPG supplements to get coverage to >98%.

Scan is used to make testing of sequential circuits tractable.

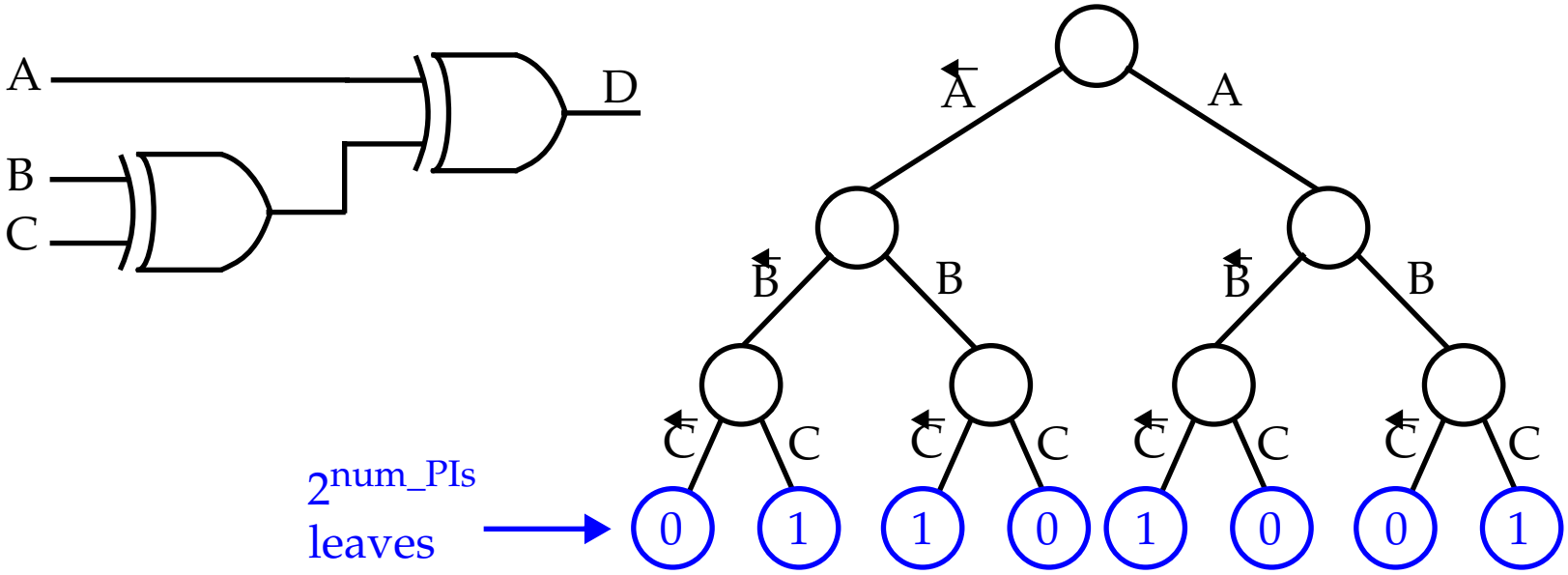
Penalties include:

- Scan hardware occupies between 5-20% of silicon area.
- Performance impact.
- Additional pins, e.g., *scan_in* and *scan_out*.
- Slower to apply.

Allows combinational ATPG to be applied to test sequential logic.

Search Space Abstractions

- Binary Search Trees.



The leaves represent the output of the good machine.

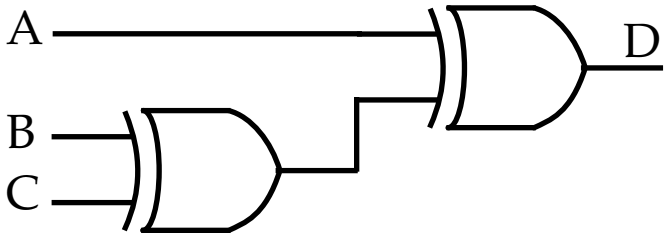
All ATPG algorithms implicitly search this tree, and in the worst case, must examine the entire tree to prove a fault is *untestable*.

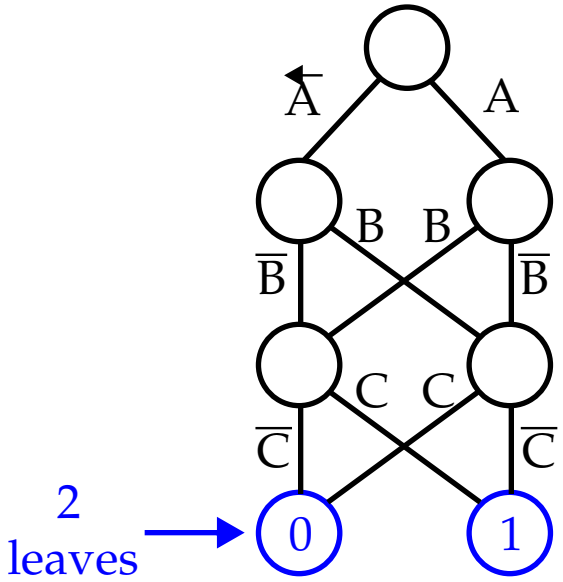
Note that *untestable* faults do not affect the circuit's logic function.

Algorithms that are able to search the entire tree are called **complete**.

Search Space Abstractions

- BDDs (Binary Decision Diagrams)





The *maxterms* and *minterms* are the product of the visited nodes.

Unfortunately, the order in which the PIs are expanded in the BDD dramatically effects the compute time of algorithms that use them.

ATPG Algebras

Boolean set notation that is capable of representing both good and faulty machines simultaneously.

Roth uses a 5-valued algebra.

Muth later showed that testing FSMs required an expansion of *X*.

Symbol	Roth's algebra		Muth's algebra	
	Good	Failing	Good	Failing
D	1	0	1	0
$\overline{D}$	0	1	0	1
0	0	0	0	0
1	1	1	1	1
X	X	X	X	X
G0	-	-	0	X
G1	-	-	1	X
F0	-	-	X	0
F1	-	-	X	1

ATPG Algorithm Types

- Exhaustive:

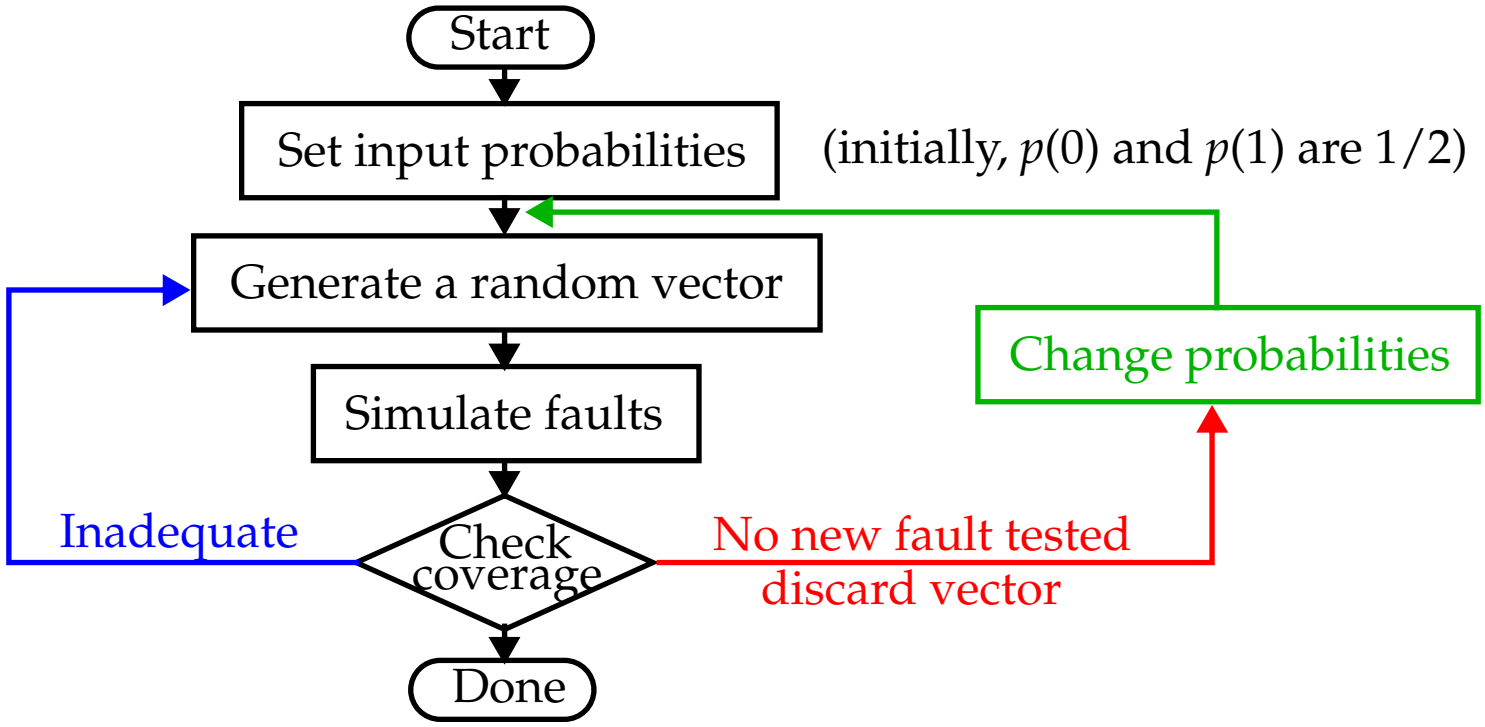
  2^n input patterns

- Random Pattern Generation (**RPG**):

  Fault simulation is essential in order to select useful patterns.

  RPG saturates at 60-80% fault coverage -- D-algo needed to improve this.

  Weighted random patterns: 0 and 1 are not equally likely.



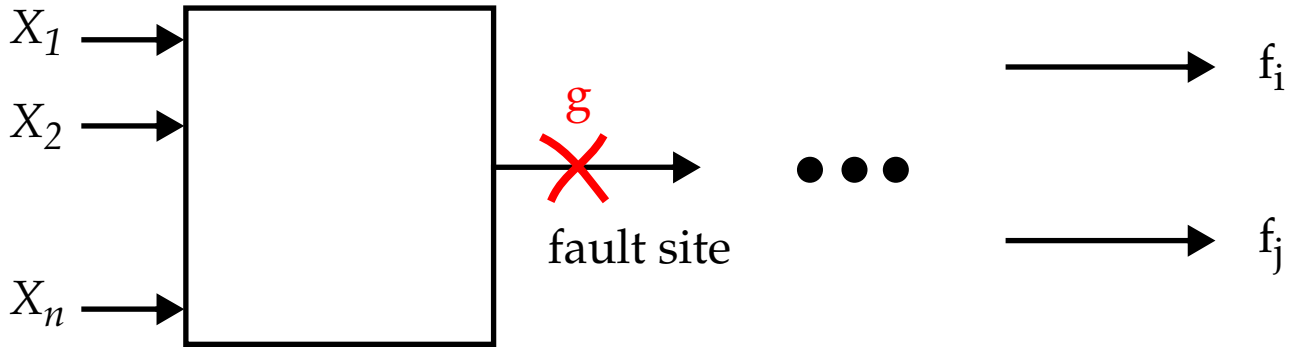
ATPG Algorithm Types

- Symbolic (Boolean Difference):

  Shannon's Expansion Theorem: a Boolean function F(X_1, X_2, ..., X_n) can be expanded about any variable, say X_2, as:

$$F(X_1, X_2, \ldots, X_n) = X_2 \cdot F(X_1, 1, \ldots, X_n) + \overline{X_2} \cdot F(X_1, 0, \ldots, X_n)$$

Let g = G(X_1, X_2, ..., X_n) represent the function at the fault site:



Let f_j = F_j(g, X_1, X_2, ..., X_n), then the Boolean difference is:

$$\frac{\partial F_j}{\partial g} = F_j(1, X_1, X_2 \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n)$$

ATPG Algorithm Types

- Symbolic (cont.):

  Fault detection requirements are expressed as:

  - $G(X_1, X_2, ..., X_n) = 1$
  - $\frac{\partial F_j}{\partial g} = F_j(1, X_1, X_2 \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) = 1$

  Due to high complexity of Boolean difference, it is not efficient for large circuits.

- Path Sensitization Methods (preferred method):

  Three steps:

  (a) **Fault Activation**: Force tested node to opposite of fault value.

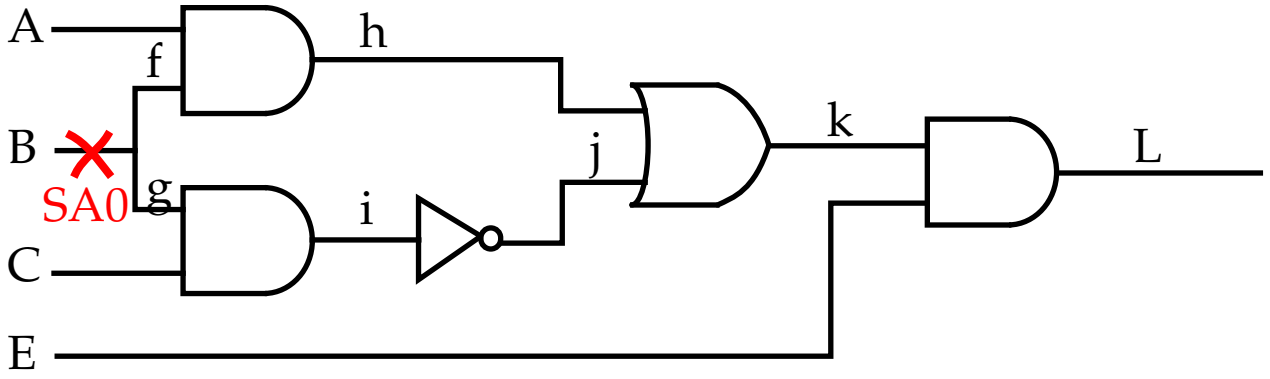
  (b) **Fault Propagation**: Also called *fault sensitization*. Propagate the effect to one or more POs.

  (c) **Line justification**: Justify internal signal assignments made to activate and sensitize faults.

ATPG Algorithm Types

- Path Sensitization Methods (cont.):

  Steps (b) and (c) may result in a conflict, i.e., different values assigned to the same signal, and require *backtracking*.



If we target *B SA0*, *fault activation* requires $B = 1$, $f = D$ and $g = D$.

Fault propagation: Three scenarios are possible, paths *f-h-k-L*, *g-i-j-k-L* and both paths.

Path *f-h-k-L* requires A=1, j=0 and E=1.

Line justification: Only j needs to be justified. Backward logic simulation requires i=1. However, g is D so its not possible -- backtrack.

ATPG Algorithm Types

- Boolean Satisfiability and Implication Graph Methods:

  x_i and $\bar{x}_i$ are literals, α_k and β_k are any two literals:

  $$\sum \alpha_k \beta_k = 0 \qquad \text{(non-tautology -- always false)}$$

  $$\prod (\alpha_k + \beta_k) = 1 \qquad \text{(satisfiability)}$$

  Objective is to find a set of assignments for the x_is that satisfy these sets of Boolean clauses.

  2-SAT problem (each clause has two literals) is solvable in polynomial time.
  3-SAT problem takes exponential time.

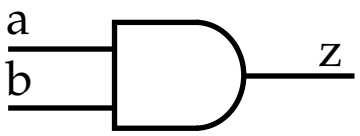
  It is possible to formulate a Boolean product-of-sums expression, that if satisfied, indicates a test for the fault.

  These algorithms are now the fastest known for huge circuits.

ATPG Algorithm Types

- Boolean Satisfiability and Implication Graph Methods (cont.):

  The Boolean function for a logic gate is captured in equations, e.g.,



  If $a = 0$ then $z = 0$

  If $b = 0$ then $z = 0$

  If $z = 1$ then $a = 1$ AND $b = 1$

  If $a = 1$ AND $b = 1$ then $z = 1$

  A **cube** is designed for each of these equations so that if the signals are consistently labeled, the cube is 0.

$$\bar{a}z + \bar{b}z + z(\overline{ab}) + ab\bar{z} = 0 \xrightarrow[\text{simplifies}]{} \bar{a}z + \bar{b}z + ab\bar{z} = 0$$

  Boolean false function:

$$F_{\text{AND}}(a, b, c) = z \oplus (ab) = \bar{a}z + \bar{b}z + ab\bar{z}$$

  Only 0 when a, b and z take values consistent with the AND function.

ATPG Algorithm Types

- Boolean Satisfiability and Implication Graph Methods (cont.):

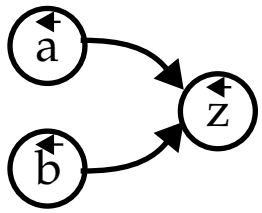
  Complement of f_{AND} is the truth expression or satisfiability.

  An efficient way to find satisfying variable assignments for false or truth functions is the implication graph.

  Boolean variable x is represented by 2 literals x and $\bar{x}$.

  If $x = 1$, x assumes a true state, if $x = 0$, $\bar{x}$ is true.

  if-then clauses can be represented with arcs from *if* literal to *then* literal:



Implication graph

Conversion to a **transitive closure graph**.

Here, if a node is set to true, e.g., $\bar{a}$, all reachable nodes are also set to true.

This allows very efficient ***global*** analysis of signal implications.

ATPG Algorithms

Algorithm	Estimated speedup over D-algorithm	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

Ibarra and Sahni in 1975 showed that ATPG is **NP-complete**, therefore no polynomial expression is known for the compute time.

These algorithms employ *heuristics* that:

- Find all necessary signal assignments for a test as early as possible.
- Search as little of the above decision space as possible (worst case is $2^{\text{num\_PIs}} * 4^{\text{num\_ffs}}$).

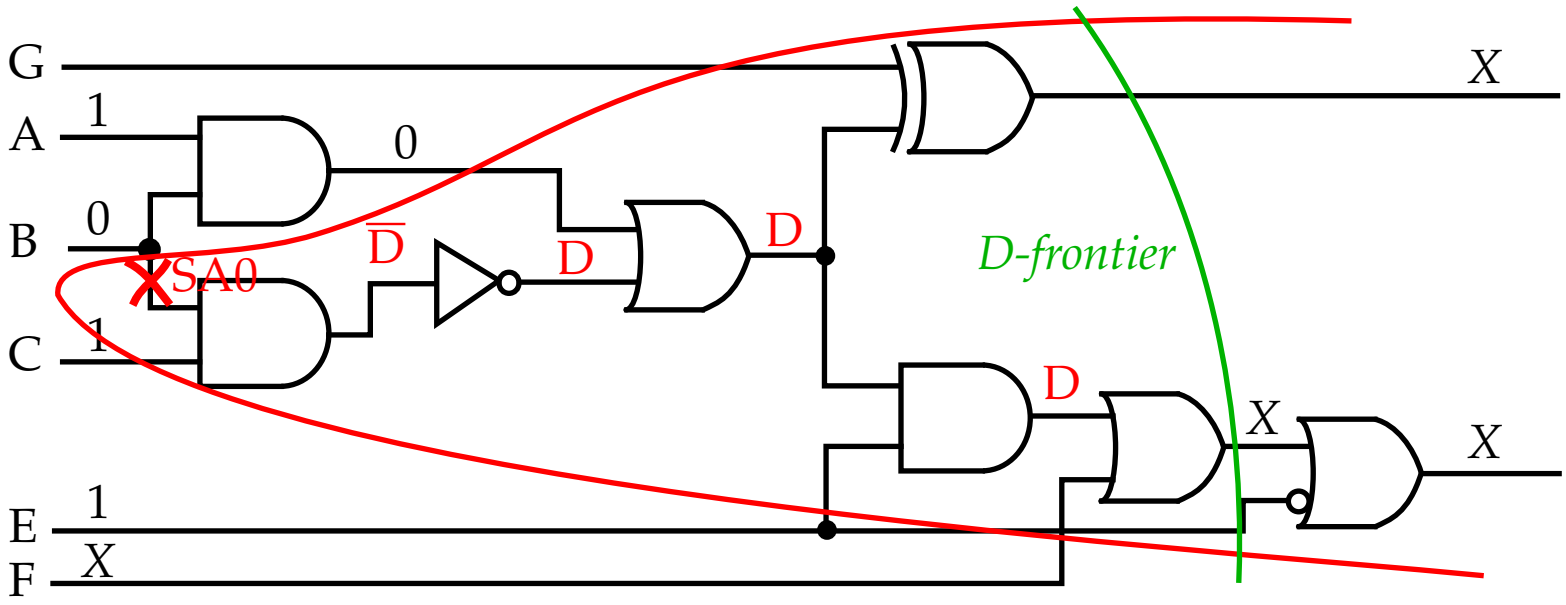
ATPG Algorithms

Since combinational fault simulation is $O(n^2)$, RPG and fault simulation is much more efficient.

This is the driver for using RPG followed by ATPG for the *hard-to-test* faults.

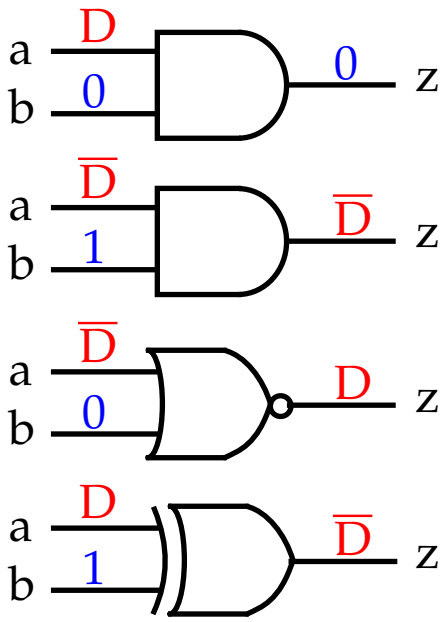
Common definitions:

- *Fault cone*: The portion of a circuit whose signals are reachable by a forward trace of the circuit topology starting at the fault site.



ATPG Algorithms

- *Forward implication*: Results when the inputs to a logic gate are labeled so that the output can be uniquely determined.



AND gate implication table

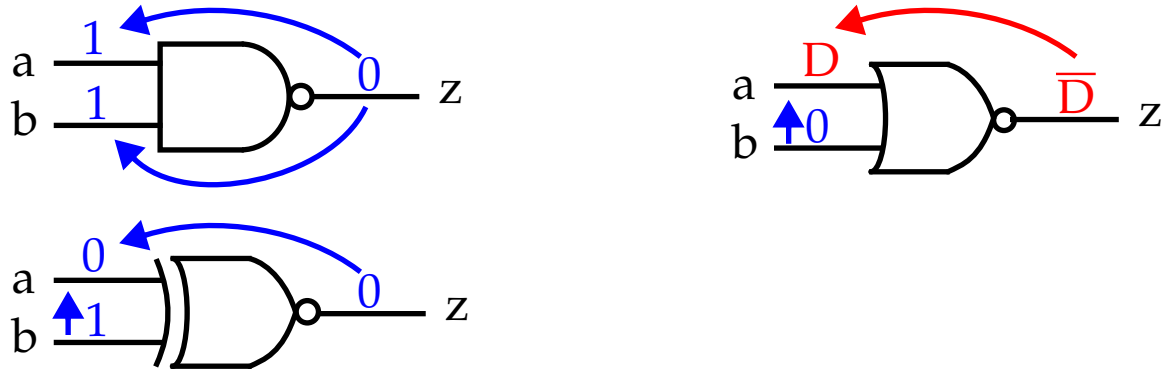
a/b	0	1	X	D	$\overline{D}$
0	0	0	0	0	0
1	0	1	X	D	$\overline{D}$
X	0	X	X	X	X
D	0	D	X	D	0
$\overline{D}$	0	$\overline{D}$	X	0	$\overline{D}$

- *Backward implication*: It is the unique determination of all inputs of a gate for a given output and possibly some of the inputs.

  Backward implication is usually implemented procedurally since tables are cumbersome for gates with more than 2 inputs.

ATPG Algorithms

- *Backward implication* (cont.):



- *Implication Stack*: Used to efficiently track that portion of the binary decision tree has already been traversed.

Here, the PIs were set in order *A*, *C*, *E* and *B*.

Also, *B* was set to 1 but failed.

Stack ptr

Signal	Value	Alternative tried
A	1	NO
C	1	NO
E	1	NO
B	0	YES

ATPG Algorithms

- *D-frontier*: The set of all gates with D or $\overline{D}$ at the inputs and X at the output. Divides the circuit into a portion with faults effects and one without.

- *Backtrack*: ATPG algorithm backtracks if:
  (a) The D-frontier becomes empty (fault effect cannot propagate further).
  (b) A signal is inconsistently assigned both 0 and 1 in order to satisfy the testing conditions.

  Alternatives are tried using the implication stack, which causes the tree to be searched in a **depth-first** fashion.

Here, 1 on F blocks fault propagation, so 0 is tried.

Also, *B* was set to 1 but failed earlier.

Stack ptr →

Signal	Value	Alternative tried
E	1	NO
B	0	YES
F	0	YES

ATPG Algorithms

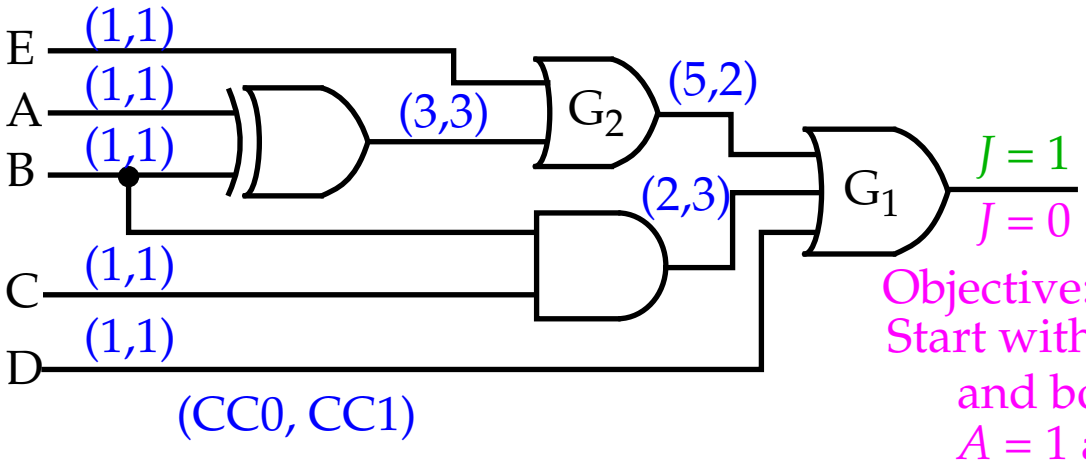
- *Objectives*: Goals to be achieved during ATPG.

  Intermediate signal assignments may make it impossible to achieve the objective.

  Many of the improvements to ATPG algorithms have focused on improving the selection of the objectives, coupled with reduction in backtracks.

- *Backtrace*: An operation designed to determine which PI should be set to achieve an objective.

  Most frequently directed by combinational controllability and observability measures.



Objective: $J = 1$
Setting $D = 1$ satisfies this objective easiest.

Objective: $J = 0$
Start with hardest:Top input of G_1 and bottom input of G_2 => $A = 1$ and $B = 1$.

ATPG Algorithms

- *Branch-and-bound search*: An efficient search procedure of binary decision trees.

  Branch involves determining which input variable will be set to what value (0 or 1).

  Bounding avoids searching large portions of the decision tree by restricting the search decision choices.

  The bounding operation is important since it avoids complete exploration.

  However, decisions about bounding often need to be made with limited information.

  Heuristics are used to bound the tree search.