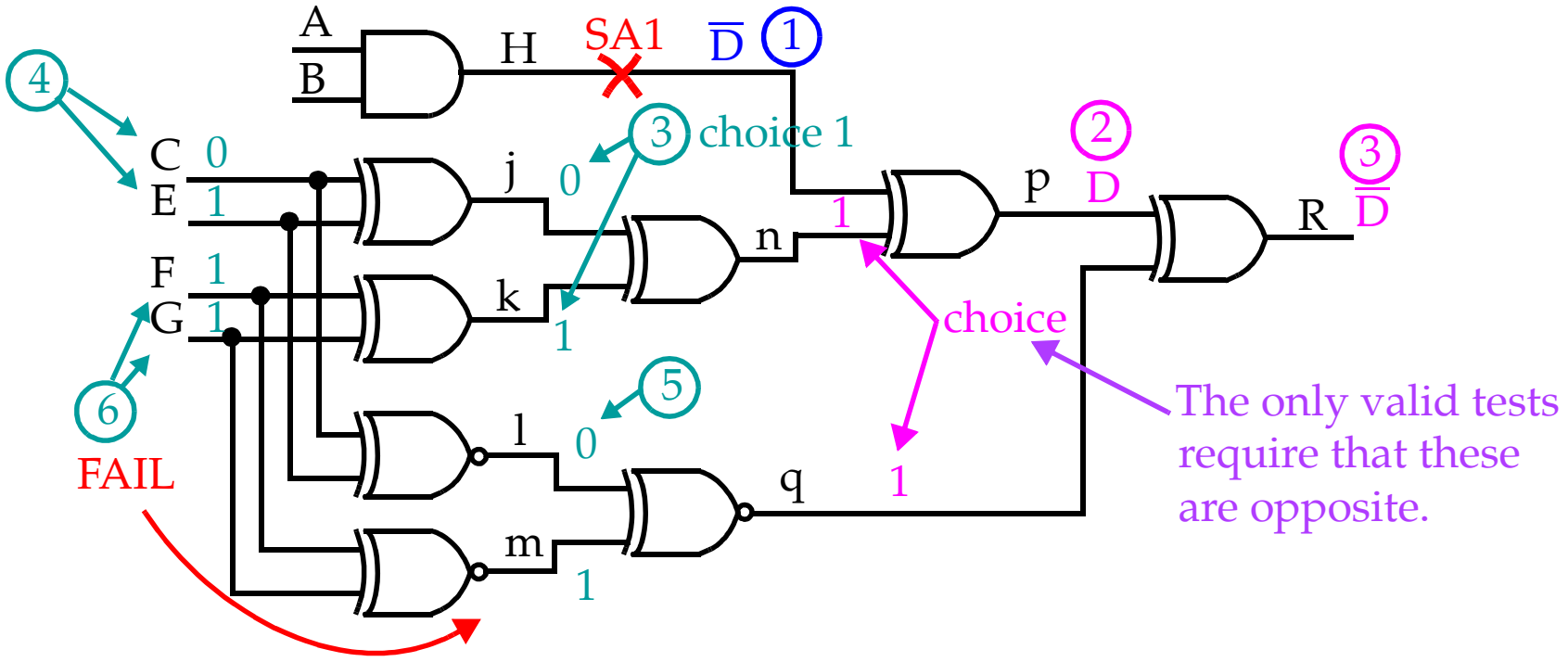
PODEM

In late '70s, IBM introduced ***error correction and translation (ECAT)*** to their DRAM to increase reliability.

D-ALG fails on attempts to generate tests for these circuits because the search is not directed.



D-ALG will eventually determine that $n = q$ is not realizable by this circuit.

PODEM

PODEM (Path-Oriented Decision-Making) introduced several standard ATPG concepts:

- PODEM expands the binary decision tree ***around the PIs*** and not around all circuit signals.

  This reduces the size of the tree from 2^n to 2^{num_PIs}.

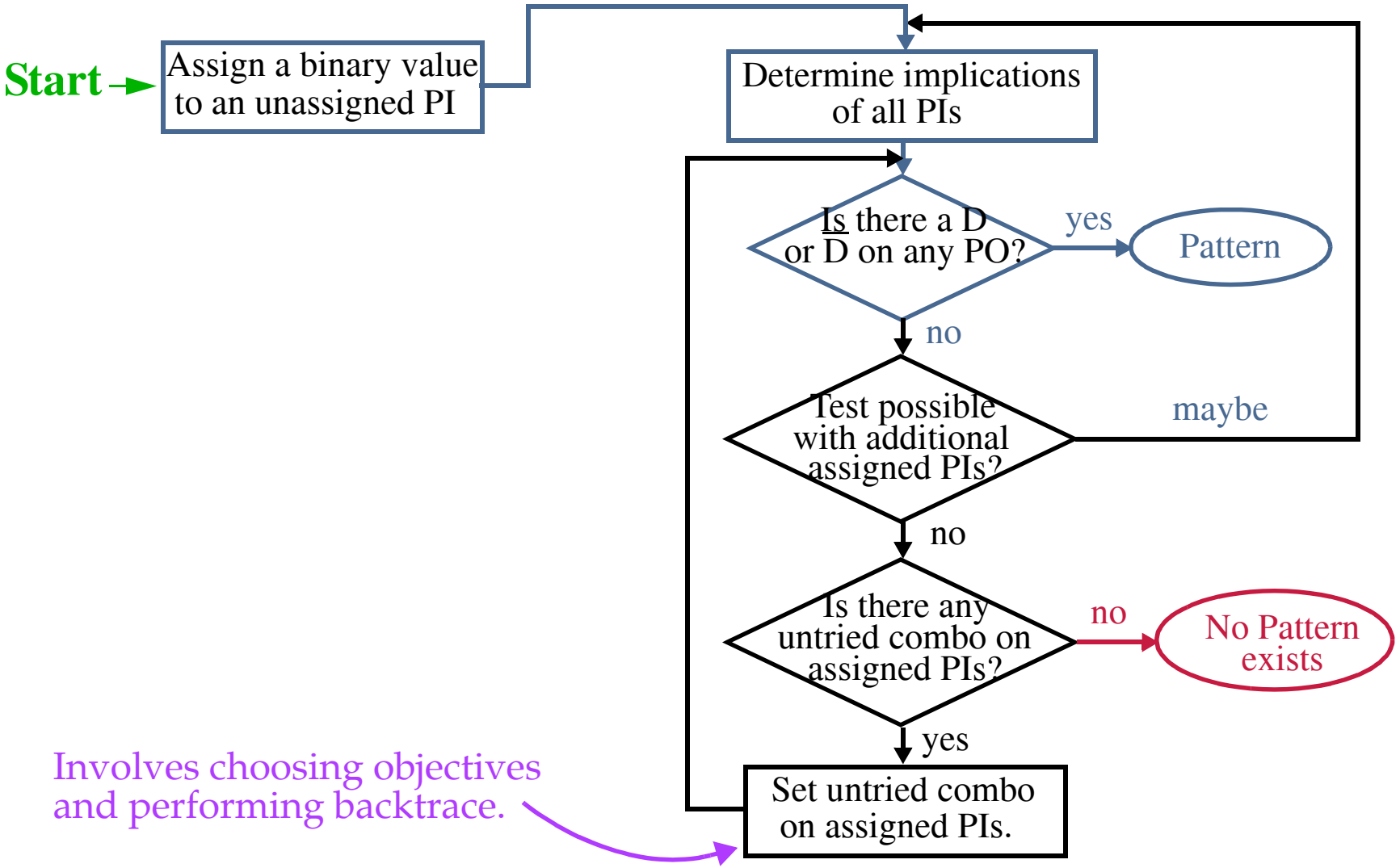
- D-ALG tended to continue intersecting ***D-cubes*** even when the ***D-frontier*** disappeared.
  PODEM introduced a subroutine to test if ***D-frontier*** still existed.

- PODEM introduced ***objectives*** and realized that choosing PIs to set was important in efficiently realizing objectives.
  Backtracing was used to obtain a PI assignment given an initial objective.

  PODEM considered the length of the path between the objective and the POs and used ***controllability measures*** to guide the ATPG algorithm.

PODEM

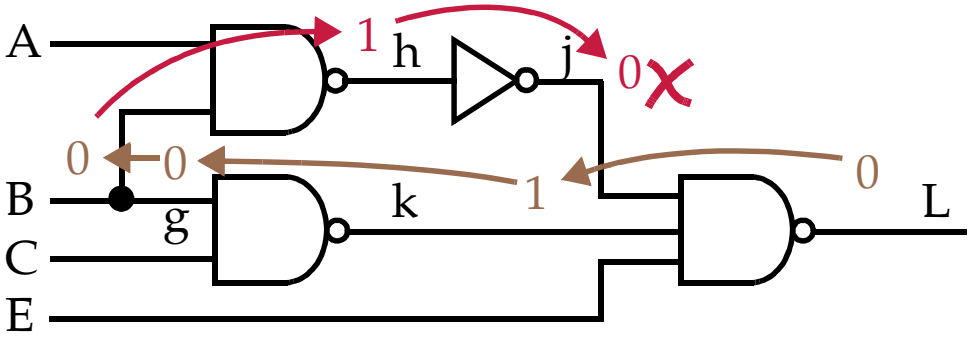
PODEM starts at the PIs instead of at ***faulty line*** like D-ALG.



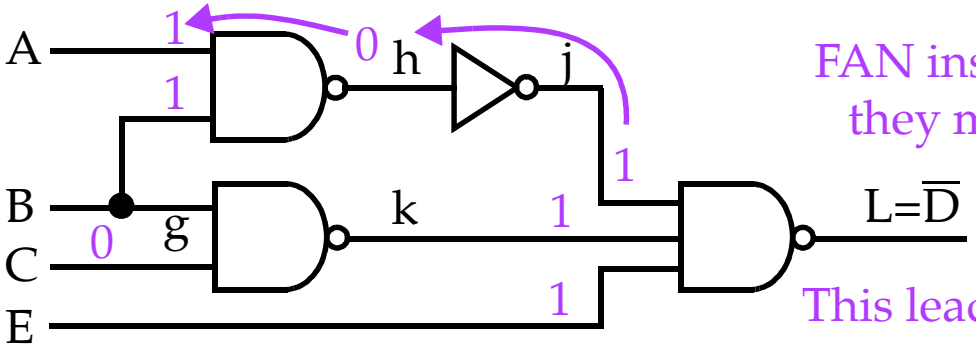
FAN

Fujiwara and Shimono introduced several novel concepts to further limit the ATPG search space and accelerate backtracing:

- ***Immediate Implications*:** PODEM misses opportunities to immediately assign values that are uniquely determined to signals.



Given objective $L = 0$, PODEM would backtrace and assign k=1, g=0 and B=0. This is unfortunate since B=1 => h=1 and j=0 which prevents objective.



FAN instead sets j, k and E to 1 since they must all be set to justify $L=\overline{D}$.

This leads to unique A and B=1, C=0.

FAN and Other Advanced ATPG Algorithms

Test book describes other novel features of the FAN algorithm.

- ***Dominator ATPG Programs***: TOPS (Kirkland and Mercer)
- ***Learning ATPG Programs***:
  - SOCRATES (Schulz et al.)
  - EST (Giraldi and Bushnell)
  - Recursive Learning (Kunz and Pradhan)
- ***Implication Graph ATPG Algorithms***:
  - NNATPG (Chakradhar et al.)
  - TRAN (Chakradhar et al.)
  - GRASP
  - NEMESIS
  - TEGUS
  - A program by Tafertshofer et al.
- ***BDD-Based ATPG Algorithms*** (performance poor on multipliers):
  - CATAPULT (Gaede et al.)
  - TSUNAMI (Stanion Bhattacharya)

Test Generation Systems

An ATPG system may contain:

- Fault generator/collapsing program
- RPG program
- Fault simulator
- ATPG program
- Test compactor

Performance criteria include:

- Fault coverage

$$\text{Fault coverage} = \frac{\text{Number of detected faults}}{\text{Total number of faults}}$$

- Fault efficiency

$$\text{Fault efficiency} = \frac{\text{Number of detected faults}}{\text{Total number of faults} - \text{Number of undetectable faults}}$$

- Vector set size
- CPU time

Test Generation Systems

SOCRATES: Starts with RPG (optionally with weighted pattern probabilities), concurrent fault simulation and fault dropping.

32 random patterns are generated in parallel and one concurrent fault simulation is carried out.

The process terminates when no faults are detected after 64 random patterns have been tried.

This is followed with several passes of ATPG.

Pass one is done usually with only 10 backtracks allowed per fault.

Each pattern is then fault simulated against all remaining faults and detected faults are dropped.

Later passes increase the number of backtracks to 50, 100 and finally 10,000.

This process outputs a ***test vector file***, a list of ***undetected faults***, a list of ***redundant faults***, a list of ***aborted faults*** and a ***backtrack distribution file***.

Test Compaction

Many ATPG systems use RPG to get 60% fault coverage, followed by ATPG.

However, many of the RPG patterns may not be as effective at providing "high" fault coverage.

At the end of ATPG, all patterns are fault simulated in ***reverse order*** of their generation.

Once fault coverage reaches 100%, the remaining RPG patterns are discarded.

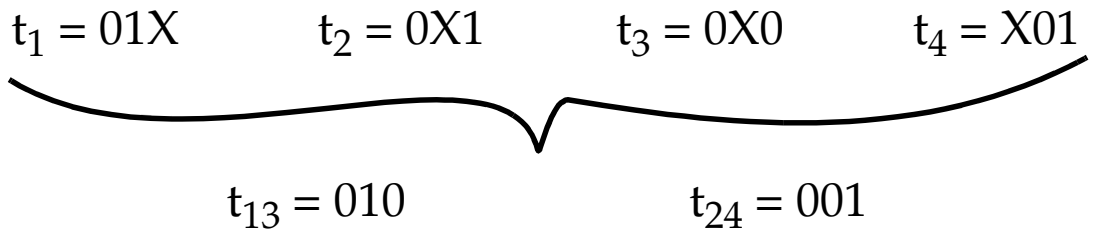
This type of ***compaction*** greatly reduces the size of the test set.

An additional ***static compaction*** method is suitable for the ATPG generated patterns, where unassigned inputs are left at *X*.

Two patterns can be combined if they are compatible, as defined by the ***D-intersection*** operator given earlier.

Test Compaction

The degree of compaction possible depends on the order in which the vectors are processed.



Optimal static compaction algorithms are impractical, so heuristic algorithms are used.

Dynamic compaction immediately assigns *1*'s and *0*'s to the unassigned PIs after the ATPG program generates them.

The ***secondary*** faults detected allows additional fault dropping.