D-Algorithm

- **D-Frontier**
  - All gates whose output values are X, but have D (or \( \overline{D} \)) on their inputs.

- **J-Frontier (Justification Frontier)**
  - All gates whose output values are known (implied by problem requirements) but are not justified by their gate inputs.
D-Algorithm

- Initialization:
  - set all line values to X
  - activate the target fault by assigning logic value to that line
- 1. Propagate D to PO
- 2. Justify all values
- `Imply_and_check()` does only necessary implications, no choices.
- if `D-alg() = SUCCESS` then return SUCCESS
  - else undo assignments and its implications
D-Algorithm vs. PODEM

**D-Algorithm**
- Values assigned to internal nodes and PI’s
- Decisions
  - Choose a D from D-Frontier
  - Assign a value to justify an output
- Conflict -- An implied value different from assigned value
- Other bounding condition
  - empty D-Frontier

**PODEM**
- Values assigned only to PI’s
- Decision
  - Choose a PI and assign a value
- No conflicts
- Bounding conditions
  - fault not excited
  - empty D-Frontier
  - X-path check
    - lookahead
PODEM

- Improvements in the original paper were:
  - **X-path check**
    - Checks that at least one gate in D-Frontier has a path to a PO such that the node values on that path are all X’s.
  - **Heuristics for selecting an objective in backtrace**
    - Among the required objectives, choose the hardest objective.
    - Among alternative objectives, choose the easiest objective.
Selection Heuristics

- hardest to control
- easiest to control
Cost Functions for Test Generation
Controllability and Observability

- Distance based.
- Fanout based.
- Probabilistic
- Many others.

\[
C_4 = f(C_1, C_2, C_3) + (3 - 1)
\]
Cost Functions for Test Generation
Controllability and Observability

- Recursive (due to Rutman)
  \[ f = 3 \]

  \[ C_0(D) = \min\{C_0(A), C_0(B), C_0(C)\} + (f - 1) \]
  \[ C_1(D) = C_1(A) + C_1(B) + C_1(C) + (f - 1) \]

- Observability of PO is set to 0

\[ O(A) = C_1(B) + C_1(C) + O(D) \]
\[ O(D) = \min\{O(D_1), O(D_2), O(D_3)\} \]
Cost Functions for Test Generation
Controllability and Observability

- SCOAP:
  - Add 1 to above definitions.
  - $C_0(PI) = 1$, $C_1(PI) = 1$.

- Probabilistic

\[
\begin{align*}
  p &= \text{probability of } 1 \quad 1 - p = \text{probability of } 0 \\
  p_D &= p_A \cdot p_B \cdot p_C \\
  O(A) &= p_B \cdot p_C \cdot O(D) \\
  p_D &= 1 - (1 - p_A)(1 - p_B)(1 - p_C) \\
  p_D &= 1 - p_A \\
  C_1(PI) &= 1/2 \\
  C_0(PI) &= 1/2 \\
  O(PO) &= 1
\end{align*}
\]
Example Circuit

Node | C0 | C1 | O
---|---|---|---
1  |   |   |   
2  |   |   |   
3  |   |   |   
4  |   |   |   
5  |   |   |   
6  |   |   |   
7  |   |   |   
8  |   |   |   

Diagram of the example circuit with nodes numbered 1 to 8.
FAN (Fanout-Oriented TG)

- Backtrace, rather than stopping at a PI, stops at an internal line called a head line.
  - A head line is a line which is fed by a fanout-free region from the PI's and whose successor is fed by signals from regions containing fanout.
  - Gives early backtrack.
FAN (Fanout-Oriented TG)

- Multiple objective backtrace
  - Multiple objectives are taken from the J-frontier
    - Example:
      - Multiple objectives are \{a=1, b=1, c=1\}
        (PODEM has only one objective)
    - These objectives are backtraced to a PI, head line, or stem.
FAN (Fanout-Oriented TG)

- Count number of requests for 1 and 0 and pick the value with highest number of requests.

```
stem
  ↓
obj. 0
  ← obj. 1
```

- Observation: FAN claims to run faster than PODEM, but no recent data substantiates this claim.
  - Head lines are often too shallow to give substantial benefits.
  - Multiple backtrace is hard to evaluate.
  - FAN also did forward and backward implications.
A gate is a dominator of a line if all paths from that line to all PO’s pass through the gate.

All off-path lines of dominator gates need to be set to non-controlling values.
All mandatory assignments are found by backward and forward implication.

- This yields a list of objectives to propagate the D forward.
- This list is then sorted in order of hardest to easiest to control. The hardest-to-control objectives are satisfied first.
Gates 1, 3, 5, and 6 are dominators of line A.

Lines B and C must be set to X/1.

Lines D and E must be set to X/1, 1/X, or X/X, depending on the parities of the paths.

Lines F and G are set to X/1 by backward implication.

Line H is set to X/1 by forward implication.

List of objectives: (B, X/1), (C, X/1), (F, X/1), (G, X/1), (D, ?), (E, ?)
Increasing Number of Mandatory Assignments

- Targeted D frontier [Niermann & Patel, HITEC]
  - Unique sensitization is really only useful when there is only one D in frontier.
  - Push all but one D of the D-frontier onto the stack, and eliminate them from the D-frontier.
  - Effectively, there is always only one D in the D-frontier.
Random Test Generation

Phase I: Random

- inexpensive $t_1$

Phase II: Deterministic

- expensive $t_2$
- total time = $t_1 + t_2$

- Is random phase useful?

[Graph showing increasing fault coverage with number of vectors, reaching 100%]

} apply deterministic
Random phase
Random Test Generation

- Some measurements show that there is very little to gain in time with two phases, if your deterministic phase is good.
For uniform distribution of 1’s and 0’s, the probability of detection of faults F s-a-0 and G s-a-1 is 1 in a million.

Having more 1’s than 0’s will increase the probability of detection of F s-a-0 but will decrease the probability for G s-a-1.

Having more 0’s than 1’s will have the opposite effect.

In general, each input may require different probability of 1’s and 0’s. [Wunderlich, DAC, 1987]
RAPS
Random Path Sensitization (Goel)

- Create random critical paths between PI’s and PO’s.
  - Select a random PO Z and a random value v.
  - Perform random backtrace with (Z, v) as the objective.
  - Continue selecting PO’s until all have assigned values.
  - Fill in any unknown PI values so as to maximize the number of critical paths.
- Results are better than random test generation.
Test Set Compaction

- Reduction of test set size while maintaining fault coverage of original test set.
  - **Static Compaction**
    - Compaction is performed after the test generation process, when a complete test set is available.
  - **Dynamic Compaction**
    - Compaction is performed as tests are being generated.
Static Compaction Techniques

- Reverse order fault simulation.
  - Remove vectors that don’t detect any faults.
- Generate partially-specified test vectors and merge compatible vectors.

\[
\begin{align*}
  t_1 &= 01x \\
  t_2 &= 0x1 \\
  t_3 &= 0x0 \\
  t_4 &= x01 \\
  t_{12} &= 011 \\
  t_3 &= 0x0 \\
  t_4 &= x01 \\
  t_{13} &= 010 \\
  t_{24} &= 001 
\end{align*}
\]

- Heuristics are used to select vectors for merging.
Dynamic Compaction Techniques

- Obtain a partially-specified test vector for the primary target fault.
- Attempt to extend the test vector to cover secondary faults.
  - Heuristics for choosing secondary faults.
COMPACTEST

- Use of independent faults for fault ordering.
- Method of maximal compaction for obtaining tests that detect large numbers of faults.
- Dynamic line justification method aimed at maximizing number of faults detected by each vector.
Independent Faults

- No test exists that detects two different faults in a set of independent faults.
- The size of the largest set of independent faults is a lower bound on the minimum test set size.
Fault Ordering

- Compute maximum independent fault sets.
  - Compute independent fault sets only within maximal fanout-free regions (FFR).

- Order faults such that largest sets of independent faults come first.

- Identify faults in FFR that can potentially be detected by the vector for a given fault.
Maximal Compaction

- Unspecify PI values that are not needed to detect target fault (heuristic).
  - Test vector may no longer detect target fault.

```
Try:

0111 → not detected
1011 → detected
1101 → detected
1110 → not detected
```

vector = 1xx1

Note that target fault is not detected by 1001
COMPACTTEST Algorithm

- Get primary target fault from front of fault list.
- Generate test for target fault that also detects other faults in its FFR if possible.
- Maximally compact the vector.
- Get secondary target fault and attempt to extend test vector to cover it.
  - If successful, try to detect additional faults in same FFR as secondary target fault, and maximally compact newly-specified parts of test vector.
  - Else set newly-specified values back to X.
- Repeat until all inputs specified or all faults tried.
- Specify all unspecified PI’s randomly.