Overview: Major ATPG Algorithms

- Definitions
  - D-Algorithm (Roth) -- 1966
    - D-cubes
    - Bridging faults
    - Logic gate function change faults
  - PODEM (Goel) -- 1981
    - X-Path-Check
    - Backtracking
  - Summary

Forward Implication

- Results in logic gate inputs that are significantly labeled so that output is uniquely determined

AND gate forward implication table:

<table>
<thead>
<tr>
<th>a</th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>d</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>d</td>
<td>x</td>
<td>0</td>
<td>d</td>
</tr>
</tbody>
</table>

Backward Implication

- Unique determination of all gate inputs when the gate output and some of the inputs are given

Implication Stack

- Push-down stack. Records:
  - Each signal set in circuit by ATPG
  - Whether alternate signal value already tried
  - Portion of binary search tree already searched

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Alternative tried</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>YES</td>
</tr>
</tbody>
</table>

Implication Stack, Decision Tree, and Backtrack
Objectives and Backtracing in ATPG

- **Objective** – desired signal value goal for ATPG
  - Guides it away from infeasible/hard solutions
  - Uses heuristics
    - E.g. which fault site to choose first?
- **Backtrace** – Determines which primary input and value to set to achieve objective
  - Use heuristics such as nearest PI
- **Forward trace** – Determines gate through which the fault effect should be sensitized
  - Use heuristics selecting output that is closest to the present fault effect

Branch-and-Bound Search

- Efficiently searches binary search tree
- **Branching** – At each tree level, selects which input variable to set to what value
- **Bounding** – Avoids exploring large tree portions by artificially restricting search decision choices
  - Complete exploration is impractical
  - Uses heuristics
- Example:
  - For a circuit with inputs A, B, C, D and E: Å̅B.... does not achieve objective.

D-Algorithm – Roth (1966)

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - D-Cube
  - D-Calculus
  - Implications – forward and backward
  - Implication stack
  - Backtrack
  - Test Search Space

Singular Cover - Example

- Minimal set of logic signal assignments to represent a function
  - Show prime implicants and prime implicates of Karnaugh map (with explicitly showing the outputs too)

D-Cube - Example

- Collapsed truth table entry to characterize logic
- Use Roth’s 5-valued algebra
- Can change all D’s to D’s and D’s to D’s (do both)
- AND gate:

<table>
<thead>
<tr>
<th>AND Gate Propagation D-Cubes</th>
<th>A</th>
<th>B</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows 3 &amp; 1</td>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>Reverse inputs</td>
<td>1</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>AND two cubes</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Interchange D and D</td>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>D-containment – Cube a contains Cube b if b is a subset of a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D-Cube Operation of D-Intersection

- ψ – undefined (same as φ)
- μ or λ – requires inversion of D and D
- **D-intersection**: \( 0 \cap 0 = 0 \cap X = X \cap 0 = 0 \)
  - \( 1 \cap 1 = \top \cap X = X \cap 1 = 1 \)
  - \( X \cap X = X \)

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Primitive D-Cube of Failure (PDF)**

- Models circuit faults:
  - Stuck-at-0
  - Stuck-at-1
  - Other faults, such as Bridging fault (short circuit)
- Arbitrary change in logic function
- AND Output sa0:  "1 1 D"
- AND Output sa1:  "0 X D"
- Wire sa0:  "D"
- *Propagation D-cube* — models conditions under which fault effect propagates through gate

**Implication Procedure**

1. Model fault with appropriate *primitive D-cube of failure (PDF)*
2. Select *propagation D-cubes* to propagate fault effect to a circuit output (*D-drive procedure*)
3. Select *singular cover cubes* to justify internal circuit signals (*Consistency procedure*)
- Put signal assignments in test cube
- Regrettably, cubes are selected very arbitrarily by D-ALG

**Bridging Fault Circuit**

![Bridging Fault Circuit Diagram](image)

**Construction of Primitive D-Cubes of Failure**

1. Make cube set $\alpha_1$ when good machine output is 1 and set $\alpha_0$ when good machine output is 0
2. Make cube set $\beta_1$ when failing machine output is 1 and $\beta_0$ when it is 0
3. Change $\alpha_1$ outputs to 0 and D-intersect each cube with every $\beta_0$. If intersection works, change output of cube to D
4. Change $\alpha_0$ outputs to 1 and D-intersect each cube with every $\beta_1$. If intersection works, change output of cube to D

**Bridging Fault D-Cubes of Failure**

<table>
<thead>
<tr>
<th>Cube-set</th>
<th>a</th>
<th>b</th>
<th>a*</th>
<th>b*</th>
<th>PDFs for Bridging fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>$\beta_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Gate Function Change D-Cube of Failure**

<table>
<thead>
<tr>
<th>Cube-set</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Cube-set</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>$\alpha_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\beta_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\beta_1$</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>$\beta_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\beta_1$</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

- PDFs for AND changing to OR
- 0 1 D
- 1 0 D
Propagation D-Cube

- Collapsed truth table entry to characterize logic
- Use Roth's 5-valued algebra
- AND gate: use the rules given earlier using $\alpha$ and $\beta$ but in this case work with good circuit only

Write all primitive Cubes of AND gate and then create propagation cubes

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

D-Algorithm – Top Level

1. Number all circuit lines in increasing level order from PIs to POs;
2. Select a primitive D-cube of the fault to be the test cube;
3. D-drive();
4. Consistency();
5. return();

D-Algorithm -- D-drive

while (untried fault effects on D-frontier)
select next untried D-frontier gate for propagation;
while (untried fault effect fanouts exist)
    select next untried fault effect fanout;
generate next untried propagation D-cube;
if (intersection fails or is undefined) continue;
if (all propagation D-cubes tried & failed) break;
if (intersection succeeded)
    add propagation D-cube to test cube -- recreate D-frontier;
Find all forward & backward implications of assignment;
save D-frontier, algorithm state, test cube, fanouts, fault;
break;
else if (intersection fails & D and $\overline{D}$ in test cube) Backtrack();
else if (intersection fails) break;
if (all fault effects unpropagatable) Backtrack();

D-Algorithm -- Consistency

g = coordinates of test cube with 1’s & 0’s;
if (g is only PIs) fault testable & stop;
for (each unjustified signal in g)
    Select highest # unjustified signal $z$ in g, not a PI;
    if (inputs to gate $z$ are both D and $\overline{D}$) break;
    while (untried singular covers of gate $z$)
        select next untried singular cover;
        if (no more singular covers)
            If (no more stack choices) fault untestable & stop;
            else if (untried alternatives in Consistency)
                pop implication stack -- try alternate assignment;
                else
                    if (singular cover D-intersects with $z$) delete $z$ from g, add inputs to singular cover to g, find all forward and backward implications of new assignment, and break;
                    If (intersection fails) mark singular cover as failed;
        else
            Backtrack();
            D-drive();
            If (singular cover D-intersects with $z$) delete $z$ from g, add inputs to singular cover to g, find all forward and backward implications of new assignment, and break;
            If (intersection fails) mark singular cover as failed;

Backtrack

if (PO exists with fault effect) Consistency();
else pop prior implication stack setting to try alternate assignment;
if (no untried choices in implication stack)
    fault untestable & stop;
else return;

Circuit Example 7.1 and Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c</td>
<td>F</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Steps for Fault $d'\text{sa}0$**

- **Step 1 – D-Drive – Set $A = 1$**
  
- **Step 2 – D-Drive – Set $f = 0$**

<table>
<thead>
<tr>
<th>Step</th>
<th>Cube type</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$d$</th>
<th>$e$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PDF of AND gate</td>
<td>1</td>
<td>1</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Prop. D-cube for NOR</td>
<td>D</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sing. Cover of NAND</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test ABC=111 detects $d'\text{sa}0$

**Example 7.2 Fault $A \text{sa}0$**

- **Step 1 – D-Drive – Set $A = 1$**

- **Step 2 – D-Drive – Set $f = 0$**

- **Step 3 – D-Drive – Set $k = 1$**

- **Step 4 – Consistency – Set $g = 1$**
Step 5 -- Example 7.2

- **Step 5 – Consistency – \( f = 0 \) Already set

```
D
C
B
A
1
D
0
D
D
```

---

Step 6 -- Example 7.2

- **Step 6 – Consistency – Set \( c = 0 \), Set \( e = 0 \)

```
D
C
B
A
1
D
0
D
D
```

---

D-Chain Dies -- Example 7.2

- **Step 7 – Consistency – Set \( B = 0 \)
- **D-Chain dies

```
D
C
B
A
1
D
0
D
D
```

Test cube: A, B, C, D, e, f, g, h, k, L

---

Example 7.3 – Fault \( s \) sa1

- **Primitive D-cube of Failure \( (2,2) \)

```
A
B
C
D
f
1
D
1
D
D
D
```

(sa,b) means that the line has CC0 = a and CC1 = b

---

Example 7.3 – Step 2 \( s \) sa1

- **Propagation D-cube for \( v \)

```
A
B
C
D
f
1
D
1
D
D
```

(sa,b) means that the line has CC0 = a and CC1 = b

---

Example 7.3 – Step 2 \( s \) sa1

- **Forward & Backward Implications

```
A
B
C
D
f
1
D
1
D
D
```

(sa,b) means that the line has CC0 = a and CC1 = b

---
Example 7.3 – Step 3 $s_{sa1}$

- **Propagation D-cube for $Z$– test**

```
\[ \begin{array}{c|cccc}
  & e & f & g & h \\
  \hline
  0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 0 \\
\end{array} \]
```

(i.b.i means that the line has $CC0 = a$ and $CC1 = b$

Example 7.3 – Fault $u_{sa1}$

- **Primitive D-cube of Failure**

```
\[ \begin{array}{c|cccc}
  & e & f & g & h \\
  \hline
  0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 0 \\
\end{array} \]
```

(i.b.i means that the line has $CC0 = a$ and $CC1 = b$

Example 7.3 – Step 2 $u_{sa1}$

- **Propagation D-cube for $v$**

```
\[ \begin{array}{c|cccc}
  & e & f & g & h \\
  \hline
  0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 0 \\
\end{array} \]
```

(i.b.i means that the line has $CC0 = a$ and $CC1 = b$

Example 7.3 – Step 2 $u_{sa1}$

- **Forward and backward implications**

```
\[ \begin{array}{c|cccc}
  & e & f & g & h \\
  \hline
  0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 0 \\
\end{array} \]
```

(i.b.i means that the line has $CC0 = a$ and $CC1 = b$

Example 7.3 – Backtrack

- **Need alternate propagation D-cube for $v$**

```
\[ \begin{array}{c|cccc}
  & e & f & g & h \\
  \hline
  0 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 0 \\
\end{array} \]
```

(i.b.i means that the line has $CC0 = a$ and $CC1 = b$

**Inconsistent**

- $d = 0$ and $m = 1$ cannot justify $r = 1$
  - (equivalence)
  - Backtrack
  - Remove $B = 0$ assignment
Example 7.3 – Step 3 \( u \) sa1

- Propagation D-cube for \( v \)

Example 7.3 – Step 4 \( u \) sa1

- Propagation D-cube for \( Z \)

PODEM* -- Goel (1981)

- New concepts introduced:
  - Expand binary decision tree only around primary inputs
  - Use X-PATH-CHECK to test whether D-frontier still there
  - Objectives -- bring ATPG closer to propagating D(\( D \)) to PO
  - Backtracing

PODEM High-Level Flow

1. Assign binary value to unassigned PI
2. Determine implications of all PIs
3. Test Generated? If so, done.
4. Test possible with more assigned PIs? If maybe, go to Step 1
5. Is there untried combination of values on assigned PIs? If not, exit: untestable fault
6. Set untried combination of values on assigned PIs using objectives and backtrace. Then, go to Step 2

IBM introduced semiconductor DRAM memory into its mainframes – late 1970’s

Memory had error correction and translation circuits – improved reliability

- D-ALG unable to test these circuits
  - Search too undirected
  - Large XOR-gate trees
  - Must set all external inputs to define output
- Needed a better ATPG tool

Motivation
Example 7.3 Again

- **Select path** $s - Y$ for fault propagation

Example 7.3 -- Step 2 $s \ sa1$

- **Initial objective:** Set $r$ to 1 to excite fault

Example 7.3 -- Step 3 $s \ sa1$

- **Backtrace from** $r$

Example 7.3 -- Step 4 $s \ sa1$

- **Set** $A = 0$ in implication stack

Example 7.3 -- Step 5 $s \ sa1$

- **Forward implications:** $d = 0$, $X = 1$

Example 7.3 -- Step 6 $s \ sa1$

- **Initial objective:** Set $r$ to 1
Example 7.3 -- Step 7
Backtrace from \( r \) again

\[
\begin{array}{c}
\text{Example 7.3 -- Step 8 \hspace{1em} s sa1} \hfill \\
\text{Set } B \text{ to 1. Implications in stack: } A = 0, B = 1
\end{array}
\]

Example 7.3 -- Step 9
Forward implications:
\( k = 1, m = 0, q = 1, Y = 1, s = D, u = D, v = D, Z = 1 \)

Backtrack -- Step 10
\( X\text{-PATH-CHECK shows paths } s - Y \text{ and } s - u - v - Z \text{ blocked (D-frontier disappeared)} \)

Step 11 -- s sa1
Set \( B = 0 \) (alternate assignment)

Backtrack -- s sa1
Forward implications:
\( d = 0, X = 1, m = 1, r = 0, s = 1, q = 0, Y = 1, v = 0, Z = 1 \). Fault not sensitized.
Step 13 -- s sa1

- Set $A = 1$ (alternate assignment)

Step 14 -- s sa1

- Backtrace from $r$ again

Step 15 -- s sa1

- Set $B = 0$. Implications in stack: $A = 1$, $B = 0$

Step 16 -- s sa1

- Forward implications: $d = 0$, $X = 1$, $m = 1$, $r = 0$. Conflict: fault not sensitized. Backtrack

Step 17 -- s sa1

- Set $B = 1$ (alternate assignment)

Fault Tested -- Step 18 s sa1

- Forward implications: $d = 1$, $m = 1$, $r = 1$, $q = 0$, $s = \overline{D}$, $v = \overline{D}$, $X = 0$, $Y = \overline{D}$
**Backtrace \((s, v)\) Pseudo-Code**

\[ v = v'_s \]

while \((s \text{ is a gate output})\)
  if \((s \text{ is NAND or INVERTER or NOR}) v = v'_s \)
  if \((\text{objective requires setting all inputs})\)
    select unassigned input \(a\) of \(s\) with hardest controllability to value \(v'_s\)
  else
    select unassigned input \(a\) of \(s\) with easiest controllability to value \(v'_s\)

\[ s = a \]

return \((s, v)\) /* Gate and value to be assigned */;

---

**Objective Selection Code**

if (gate \(g\) is unassigned) return \((g, v)\);
select a gate \(P\) from the D-frontier;
select an unassigned input \(l\) of \(P\);
if (gate \(g\) has controlling value)
  \(c = \text{controlling input value of } g\);  
else if (0 value easier to get at input of XOR/EQUIV gate)
  \(c = 1\);
else \(c = 0\);
return \((l, c)\);

---

**PODEM Algorithm**

while (no fault effect at POs)
  if \((\text{xpathcheck} \text{ (D-frontier)}) \(l, v_l\) = Objective \((\text{fault}, v_{\text{fault}})\);
    \((p_i, v_{p_i}) = \text{Backtrace} \((l, v_l)\);
    \text{Imply} \((p_i, v_{p_i})\); 
    if \((\text{PODEM} \text{ (fault}, v_{\text{fault}}) = \text{SUCCESS}) \text{ return (SUCCESS)};
    \((p_i, v_{p_i}) = \text{Backtrace} \();
    \text{Imply} \((p_i, v_{p_i})\);
    if \((\text{PODEM} \text{ (fault}, v_{\text{fault}}) = \text{SUCCESS}) \text{ return (SUCCESS)};
    \text{Imply} \((p_i, v_{p_i})\);
  else if (implication stack exhausted)
    return (FAILURE);
else Backtrack \();
return (SUCCESS);

---

**Summary**

- **D-ALG** – First complete ATPG algorithm
  - **D-Cube**
  - **D-Calculus**
  - **Implications** – forward and backward
  - **Implication stack**
  - **Backup**

- **PODEM**
  - Expand decision tree only around PIs
  - Use X-PATH-CHECK to see if D-frontier exists
  - **Objectives** -- bring ATPG closer to getting
    - D \((\delta)\) to PO
    - **Backtracing**