D Algorithm Combinational

BY

Saumya Shah (20171193)
Abhinav Navnit (2018122001)

D Algorithm

- The D algorithm was developed by Roth at IBM in 1966 and was the first complete test pattern algorithm designed to be programmable on a computer.
- It is a deterministic ATPG method for combinational circuits, guaranteed to find a test vector if one exists for detecting a fault.
- To be precise, it is a derivative of the path sensitization method in a more systematic approach that is suitable for computer acceleration
- It uses cubical algebra for the automatic generation of tests. Three types of cubes are considered :
    - Singular cube
    - Primitive D-cube of a fault (PDCF)
    - Propagation D-cube (PDC)

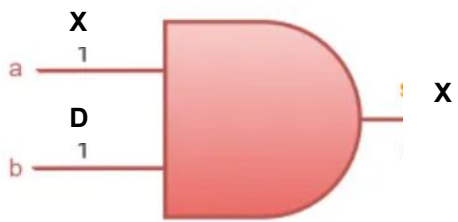
D Algebra

- The D-algebra is a 5-value logic consisting of logic: **1, 0, D, D', X**. The **D** stands for Discrepancy as in the path sensitization method. In our implementation, we took **D’** as **E.**
- The algebra relies heavily on intersection of the cubes and following algebraic rules are applicable for intersection:
    - $0 \cap 0 = 0 \cap x = x \cap 0 = 0$
    - $1 \cap 1 = 1 \cap x = x \cap 1 = 1$
    - $x \cap x = x$
    - $1 \cap 0 = D$
    - $0 \cap 1 = D’$
- As seen from the above expressions, D algebra does not follow the general boolean properties like commutativity

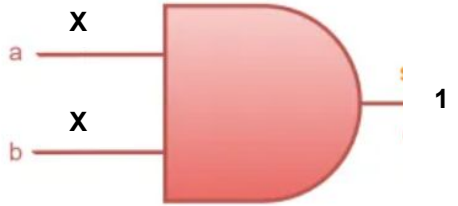
Frontiers

D algorithm contains two essential components known as frontiers. They are :

1. **D-frontier:** A set of gates whose output value is currently unknown, but have one or more **D** (or **D'**) at their inputs.



2. **J-frontier**: A set of gates whose output value is assigned, but input values have not been decided yet.



Cubes

1. Singular cover (SC)
    - the compact form of truth-table of any logic gate done using don’t cares (x).
2. Primitive D-cube of a Fault (PDCF)
    - specifies the minimum input conditions at inputs of a gate to produce an error at the output.
    - used for fault activation
3. Propagation D-cubes (PDCs)
    - causes the output of the gate to depend upon the minimum number of its specified inputs.
    - used to propagate **D** or **D’** from a specified input to the output.
4. Test cube (TC)
    - represents the signal values at various nodes in the circuit during each step of the test process.
    - contains primary inputs as well as internal nodes.

Intersection table

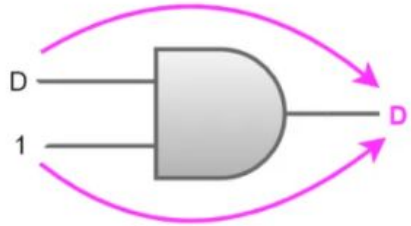
∩	**0**	**1**	**X**	**D**	**D’**
0	0	C	0	C	C
1	C	1	1	C	C
X	0	1	X	D	D’
D	C	C	D	D	X
D’	X	X	D’	X	D’

Where C is for conflict - inconsistency

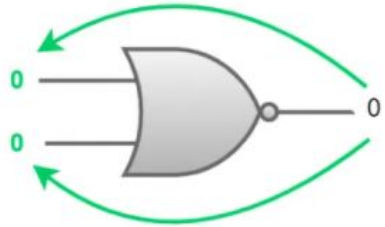
Implications

Implication means determining unknown values in input/output wires with a known set of values. In D algorithm, implication is performed whenever any decision is made.

1. **Forward Implication** refers to determining the output values from partially (or fully) specified input values of a gate.



2. **Backward Implication** refers to determining the un-specified input values from the specified output values of a gate.



Test generation process

Step 1: Fault activation

- PDCF is generated for the given fault and D frontiers formed.

Step 2: Fault propagation

- Fault is driven forward by successively intersecting the current TC with the PDC of successive gates and the new TC is obtained. Aim is to propagate the fault to output of the circuit.

Step 3: Justification

- For each J frontier formed, the input values are justified while driving back towards the primary inputs such that a consistent set of circuit input values is obtained.

Step 4: Backtrack and repeat

- If any inconsistency found, last decision is recalled and changed. Same steps are repeated for the new decision

Advantages

- It is a complete ATPG algorithm, hence it guarantees to generate a pattern for a testable fault.

Disadvantages

- All the internal signals need to be assigned in the D algorithm, thus it has a huge search space. Therefore, it is a time-consuming and slow algorithm.
- Backtracking is required for consistency checks. Backtracking becomes much more difficult, especially for reconvergent fanout circuits.

Implementation Details

- Programming Language: C++
- Library Used: STL (Standard Template Library)
- Data Structures: Graph, Vectors
- Commands to run the code
  - g++ <filename>.cpp
  - ./a.out

Inputs

A graph describing the combinational circuit

- Each Gate, PIs, POs and branch correspond to vertices of graph
- Each net corresponds to edge of the graph

Gate type of each vertex

- Various gates, PIs, POs and branches are assigned a unique number in order to identify their functionalities.

Outputs

For s-a-0 and s-a-1 fault at each net of the input circuit,

- Whether fault is testable or not
- Test Vector and Fault to PO path for testable faults

Function Details

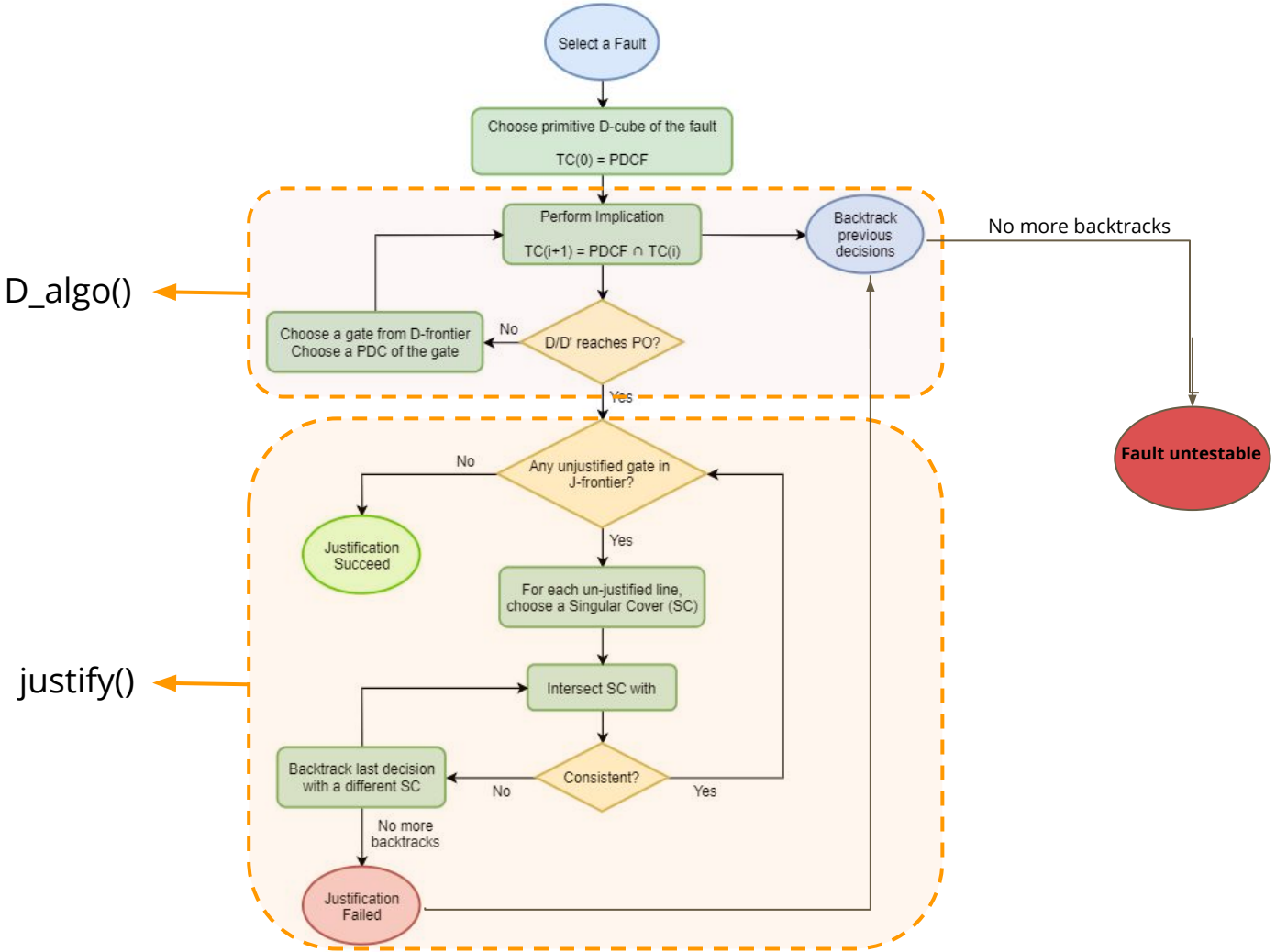
dfunct(): Returns the D frontiers when TC is given as input

jfunct(): Returns the J frontiers when TC is given as input

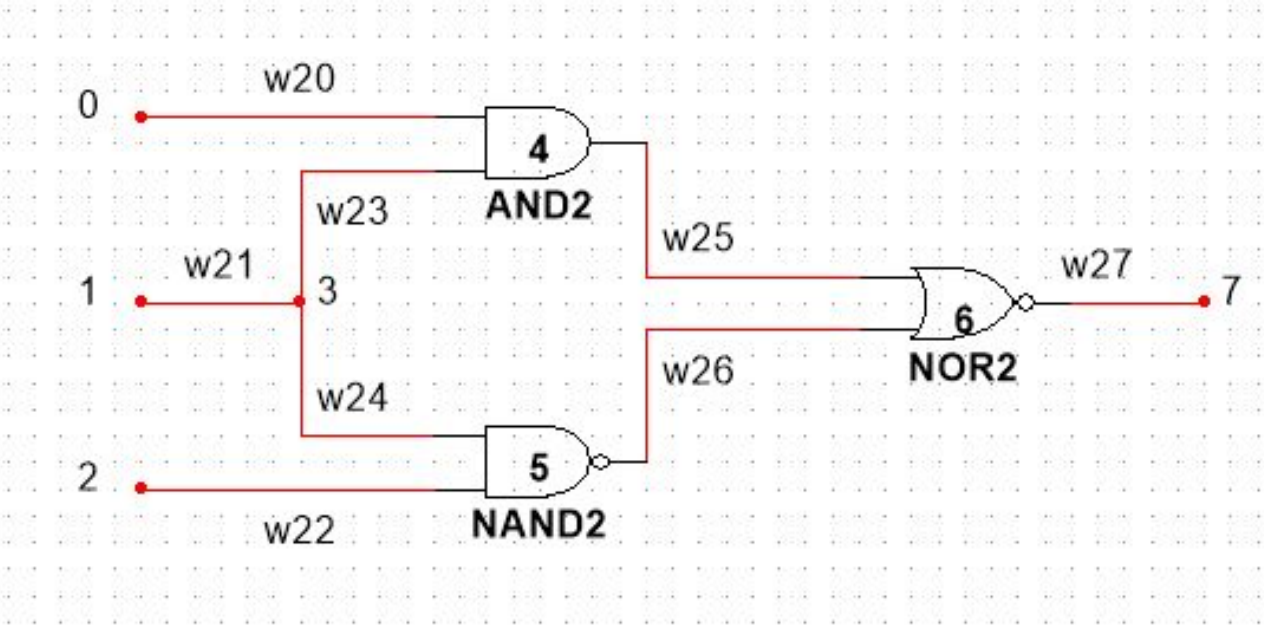
justify(): Backpropagation and Justification

D_algo(): Forward Propagation and returns the final result.

Flow


Select a Fault
Choose primitive D-cube of the fault
TC(0) = PDCF
D_algo()
Perform Implication
TC(i+1) = PDCF ∩ TC(i)
Backtrack previous decisions
No more backtracks
Choose a gate from D-frontier
Choose a PDC of the gate
No
D/D' reaches PO?
Yes
Fault untestable
No
Any unjustified gate in J-frontier?
Justification Succeed
Yes
For each un-justified line, choose a Singular Cover (SC)
justify()
Intersect SC with
Backtrack last decision with a different SC
Consistent?
No
Yes
No more backtracks
Justification Failed

Circuits under test: 1

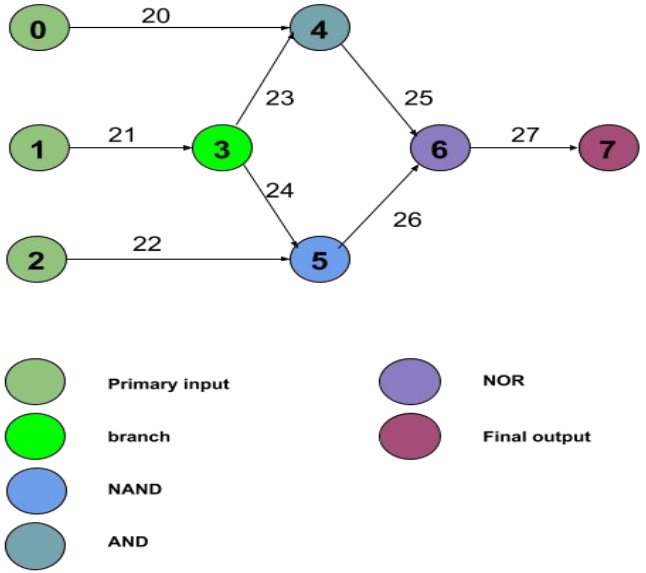


Input

Adjacency list

0	1	2	3	4	5	6
4,20	3,21	5,22	4,23 5,24	6,25	6,26	7,27

Circuit Converted to the Graph



Output

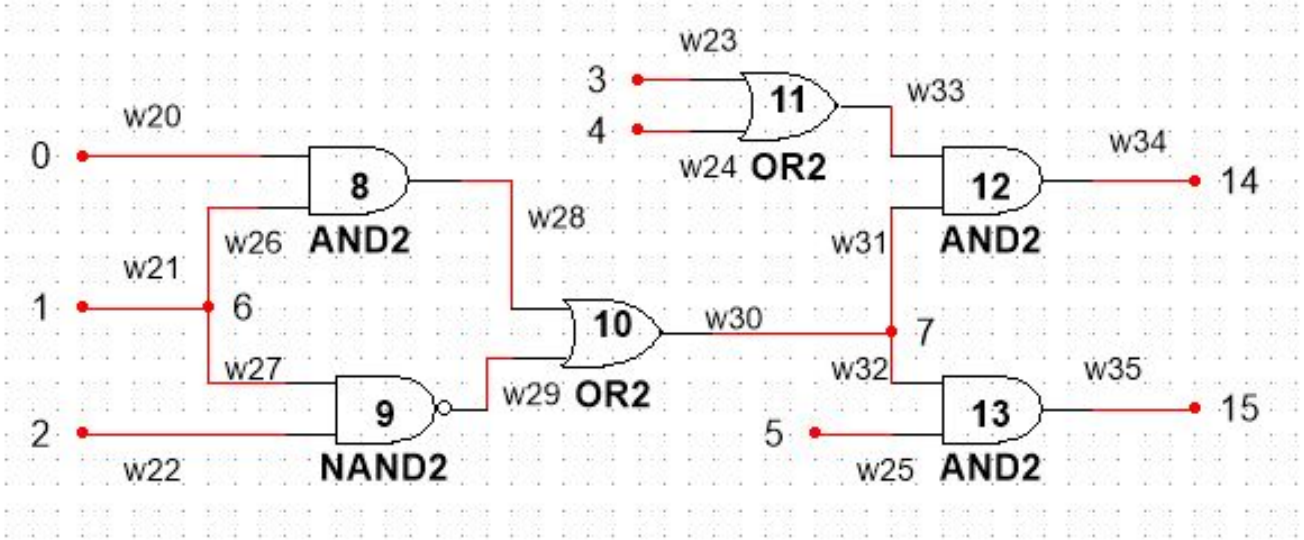
```
s-a-0 Fault at w20
TEST COMPLETED
Test Vector: 1 1 1
PATH (EdgeNumber-[GateNumber]): 20-[4]-25-[6]-27-[7]-
s-a-0 Fault at w21
TEST COMPLETED
Test Vector: 0 1 1
PATH (EdgeNumber-[GateNumber]): 21-[3]-24-[5]-26-[6]-27-[7]-
s-a-0 Fault at w22
TEST COMPLETED
Test Vector: 0 1 1
PATH (EdgeNumber-[GateNumber]): 22-[5]-26-[6]-27-[7]-
s-a-0 Fault at w23
TEST COMPLETED
Test Vector: 1 1 1
PATH (EdgeNumber-[GateNumber]): 23-[4]-25-[6]-27-[7]-
s-a-0 Fault at w24
TEST COMPLETED
Test Vector: 0 1 1
PATH (EdgeNumber-[GateNumber]): 24-[5]-26-[6]-27-[7]-
s-a-0 Fault at w25
TEST COMPLETED
Test Vector: 1 1 1
PATH (EdgeNumber-[GateNumber]): 25-[6]-27-[7]-
s-a-0 Fault at w26
TEST COMPLETED
Test Vector: 0 0 x
PATH (EdgeNumber-[GateNumber]): 26-[6]-27-[7]-
s-a-0 Fault at w27
TEST COMPLETED
Test Vector: x x x
PATH (EdgeNumber-[GateNumber]): 27-[7]-
```

```
s-a-1 Fault at w20
TEST COMPLETED
Test Vector: 0 1 1
PATH (EdgeNumber-[GateNumber]): 20-[4]-25-[6]-27-[7]-
s-a-1 Fault at w21
TEST COMPLETED
Test Vector: 0 0 1
PATH (EdgeNumber-[GateNumber]): 21-[3]-24-[5]-26-[6]-27-[7]-
s-a-1 Fault at w22
TEST COMPLETED
Test Vector: 0 1 0
PATH (EdgeNumber-[GateNumber]): 22-[5]-26-[6]-27-[7]-
s-a-1 Fault at w23
TEST COMPLETED
Test Vector: 1 0 1
PATH (EdgeNumber-[GateNumber]): 23-[4]-25-[6]-27-[7]-
s-a-1 Fault at w24
TEST COMPLETED
Test Vector: 0 0 1
PATH (EdgeNumber-[GateNumber]): 24-[5]-26-[6]-27-[7]-
s-a-1 Fault at w25
TEST COMPLETED
Test Vector: x 0 1
PATH (EdgeNumber-[GateNumber]): 25-[6]-27-[7]-
s-a-1 Fault at w26
TEST COMPLETED
Test Vector: 0 1 1
PATH (EdgeNumber-[GateNumber]): 26-[6]-27-[7]-
s-a-1 Fault at w27
TEST COMPLETED
Test Vector: x x x
PATH (EdgeNumber-[GateNumber]): 27-[7]-
```

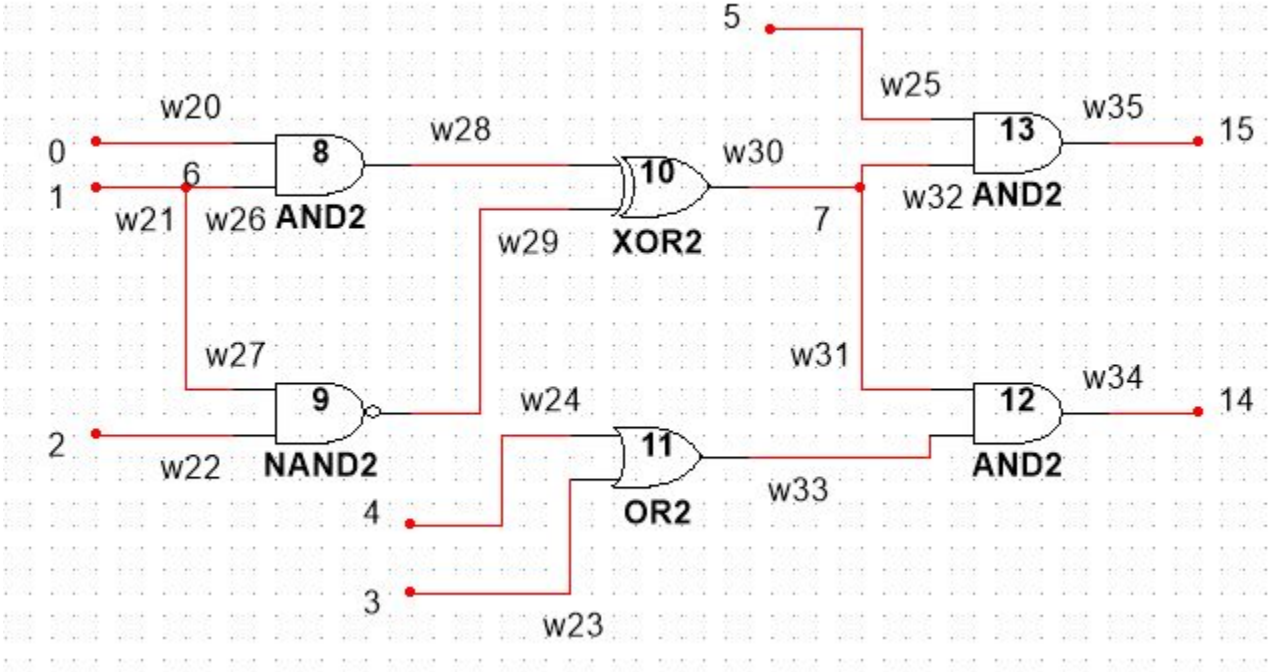
Future Work

- Extending code to work for circuits with gates having more than 2 inputs
- Automation of converting combinational circuit netlist to a graph

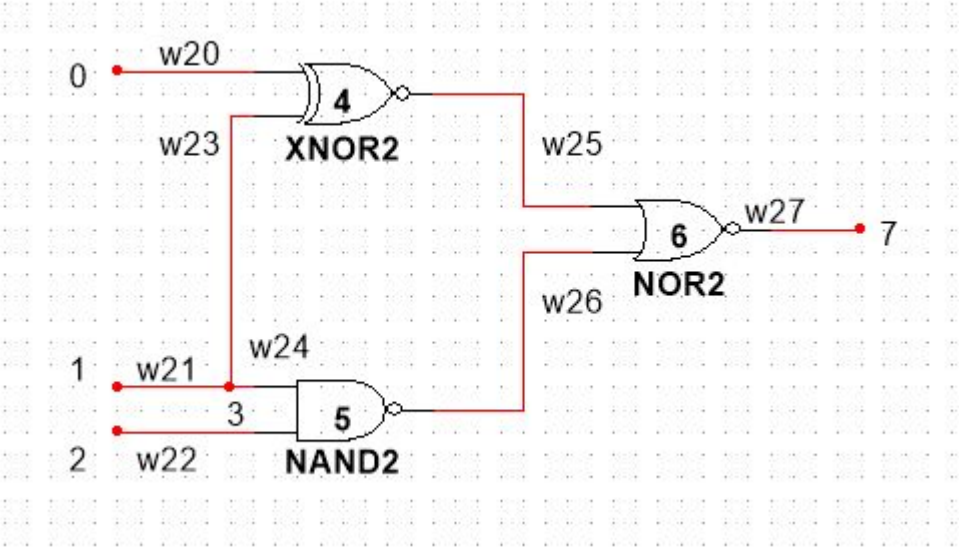
Circuits under test: 2



Circuit under test: 3



Circuit under test: 4



Circuit under test: 5