Boolean Algebra (a.k.a. Boolean Logic)

- A family of operations on a set of two symbols or values ({0,1} or {TRUE,FALSE}, or …) that obey certain laws
- Basic Operations
  - Conjunction / AND : $x \wedge y$
  - Disjunction / OR : $x \vee y$
  - Negation / Complement / NOT : $\neg x$
- Complex operations: Any formula that can be composed of basic operations
  - Exclusive OR / XOR : $x \oplus y$
  - Implication : $x \rightarrow y$
- Operations obey certain laws or axioms
- Analogy to the algebra of real numbers
  - $(B, \vee, \wedge, \neg, 0, 1) \leftrightarrow (\mathbb{R}, +, *, -, 0, 1)$

Boolean Operators as Set Operations

- The operators of Boolean algebra can also be interpreted in terms of sets

Boolean operators

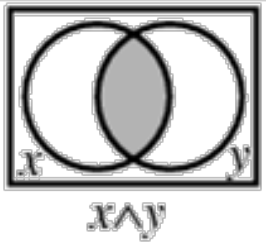
x \ y	$\wedge$ 0	$\wedge$ 1
0	0	0
1	0	1

x \ y	$\vee$ 0	$\vee$ 1
0	0	1
1	1	1

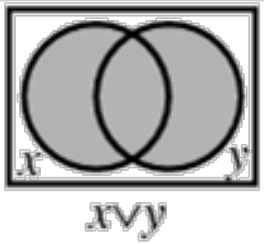
x \ y	$\rightarrow$ 0	$\rightarrow$ 1
0	1	1
1	0	1

x \ y	$\oplus$ 0	$\oplus$ 1
0	0	1
1	1	0

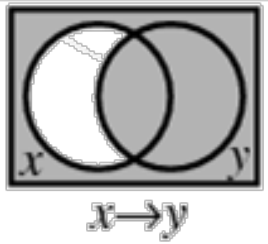
Set interpretation



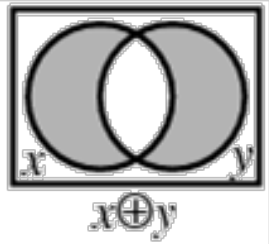
$x \wedge y$



$x \vee y$



$x \rightarrow y$



$x \oplus y$

Basic Laws of Boolean Algebra

Law	Description
Commutativity	$x \vee y = y \vee x$ $x \wedge y = y \wedge x$
Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$ $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Distributivity	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Identity	$x \vee 0 = x$ $x \wedge 1 = x$
Annihilation	$x \wedge 0 = 0$ $x \vee 1 = 1$
Idempotence	$x \vee x = x$ $x \wedge x = x$
Absorption	$x \wedge (x \vee y) = x$ $x \vee (x \wedge y) = x$

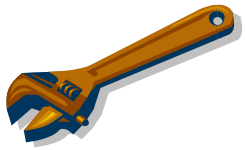
Basic Laws of Boolean Algebra

Law	Description
Complementation	$x \wedge \neg x = 0$ $x \vee \neg x = 1$
Double Negation	$\neg\neg x = x$
De Morgan	$(\neg x) \wedge (\neg y) = \neg(x \vee y)$ $(\neg x) \vee (\neg y) = \neg(x \wedge y)$

- Duality principle
  - Boolean algebra is unchanged when 0,1 and $\wedge$, $\vee$ are interchanged

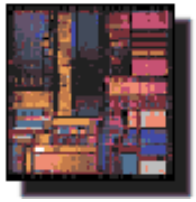






ECE 595Z
Digital Systems Design Automation

Module 2 (Lectures 3-5) : Advanced Boolean Algebra
Lecture 4



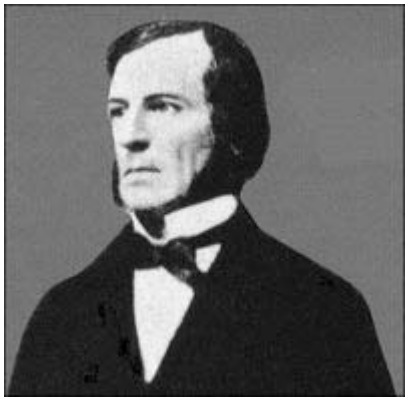
Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

Did You Know?

- Boolean logic was the invention of George Boole (1815-1864), an English mathematician and philosopher
- Published his first paper at the age of 24
- Landmark papers
  - "The mathematical analysis of logic", published in 1847.
  - "An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities", published in 1854
- Argued that there was a strong analogy between logic (then considered a discipline of philosophy) and mathematics
- Initially, his theory was ignored or criticized by the academic community
- Followed up later by a young student at MIT – for his M.S thesis in 1937
  - Showed how to use Boolean logic to optimize electromechanical relay networks



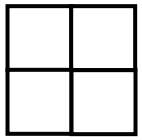
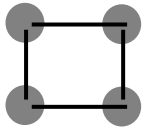
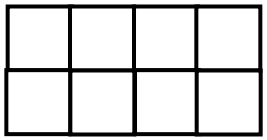
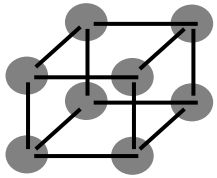
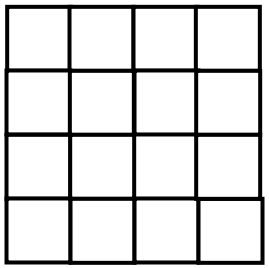
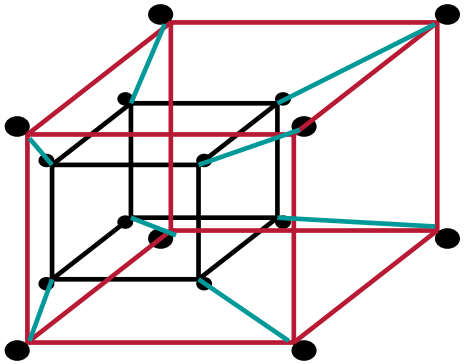
- Boole's life was tragically cut short when he died at the age of 49, at the peak of his intellectual abilities
- After walking 2 miles through a drenching rain to get to class and then lecturing in wet clothes, Boole caught a 'feverish cold'
- Boole was put to bed by his wife, Mary Everest Boole, who dumped buckets of water on him based on the theory that the remedy for an illness ought to bear resemblance to its cause

That was easy … what else?

- Quite a bit!
  - Boolean functions
  - Operations on Boolean functions
  - Representation of Boolean functions
  - Co-factors and their applications

Boolean Spaces

- Boolean space of n variables is the set of all possible values that the variables can assume
- Can be represented as an n-dimensional unit hypercube

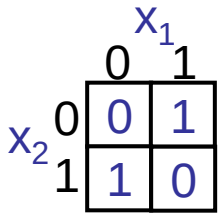
Boolean space	Karnaugh Maps	Boolean Hypercubes
$B^1 = \{0,1\}$		
$B^2 = B \times B = \{00,01,10,11\}$		
B^3		
B^4		

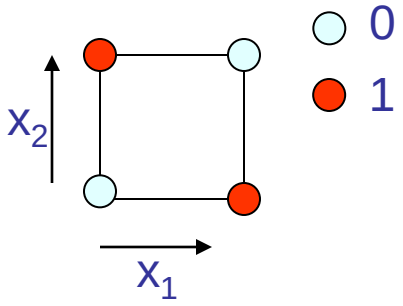
For information on hypercubes: http://en.wikipedia.org/wiki/Hypercube

Boolean Functions

- Boolean function
  - $f(x): B^n \rightarrow B$
- $x = x_1, x_2, \ldots x_n$ are variables
  - $x_i \in B$
- On-set of f
  - $\{x \mid f(x) = 1\} = f^1 = f^{-1}(1)$
- Off-set of f
  - $\{x \mid f(x) = 0\} = f^0 = f^{-1}(0)$
- Boolean functions are also called Logic functions

Example: $f(x): B^2 \rightarrow B$



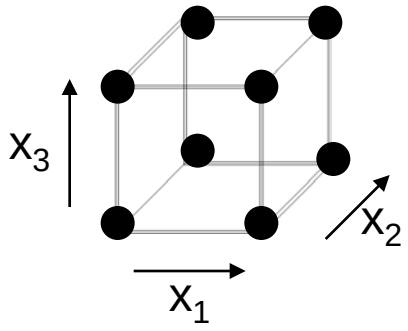


On-set : {01, 10}
Off-set: {00, 11}

Boolean Functions (contd.)

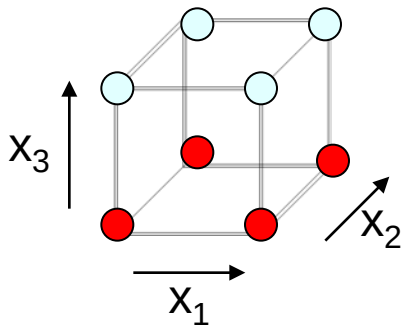
- If $f^1 = B^n$, *i.e.*, f(x) = 1, f is a tautology
- If $f^0 = B^n$, *i.e.*, f(x) = 0, f is not satisfiable
- If f(x) = g(x) for all $x \in B^n$, then f and g are equivalent

- Question: How many distinct logic functions of n variables exist?
  - Hint: Think of how many ways you can color the vertices of a Boolean cube with two colors



The Set of Boolean Functions

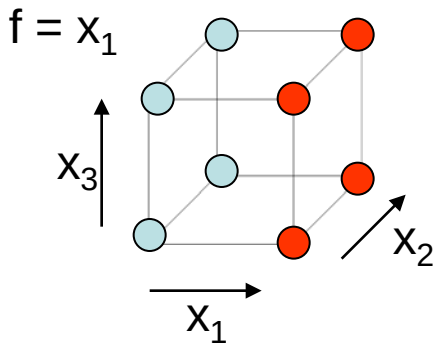
- There are 2^n vertices in input space B^n $\rightarrow$ 2^{2^n} distinct logic functions.
  - Assigning each distinct subset of vertices as the on-set results in a distinct logic function $f^1 \subseteq B^n$

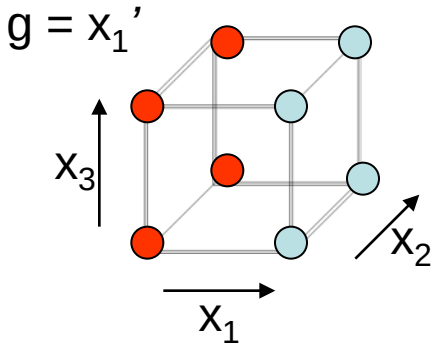


$x_1x_2x_3$	f
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

The Set of Boolean Functions

- Another way to think about Boolean functions
  - Compose them using simple functions and operators
- Simple functions
  - Constant functions (f = 0, f = 1)
  - Literals
    - A literal is a variable (x_1) or its complement (x_1')
    - Literal x_1 represents the logic function f = {x| x_1 = 1}
    - Literal x_1' represents the logic function g = {x| x_1 = 0}





Notation: $x_1' = \overline{x}_1$

Operations on Boolean Functions

Given two Boolean functions:

$f : B^n \rightarrow B$

$g : B^n \rightarrow B$

Interpretation in terms of on-set and off-set?

- AND operation

  $f \cdot g = \{x \mid f(x)=1 \wedge g(x)=1\}$

- The OR operation

  $f + g = \{x \mid f(x)=1 \vee g(x)=1\}$

- The COMPLEMENT operation (f')

  $f' = \{x \mid f(x) = 0\}$

The Algebra of Boolean Functions

- The set of all Boolean Functions together with the operations on them also satisfy the laws of Boolean Algebra

Representations of Boolean Functions

- Truth Table
- Hypercube
- Boolean Formula
  - Sum of Products (SOP), Disjunctive Normal Form (DNF), List of Cubes
  - Product of Sums, Conjunctive Normal Form (CNF), List of Conjuncts
- Network (graph) of Boolean primitives
- Binary Decision Tree, Binary Decision Diagram (BDD)

Representations of Boolean Functions

- Important questions to ask of any representation
  - Scalable (can represent large functions)?
  - Canonical?
    - If two functions are the same, then their representations are isomorphic (identical)
  - Efficient to manipulate?

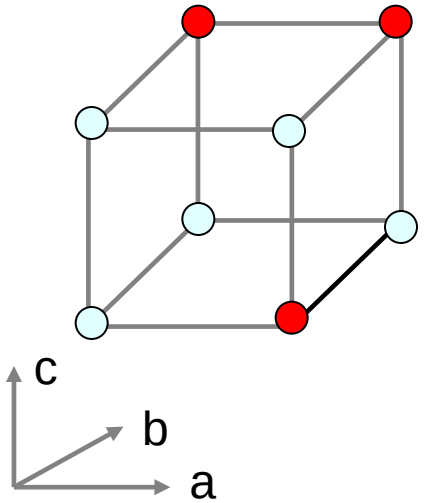
Truth Table

- Truth table of a function f : $B^n \rightarrow B$ is a tabulation of its values at each of the 2^n vertices of B^n
- The truth table representation is
  - + Canonical
  - - Intractable for large *n*

Example:

a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

f = b c + a b' c'



Boolean Formula

- Boolean functions can be represented by formulae defined as well-formed sequences of
  - Literals: x_1, x_1'
  - Boolean operators: + (OR), . (AND), ' (NOT)
    - NOT: f' = h such that $h^1 = f^0$
    - AND: (f AND g) = h such that h^1 = {x | f(x) = 1 and g(x) = 1}
    - OR: (f OR g) = h such that h^1 = {x | f(x) = 1 or g(x) = 1}
  - Parentheses: ( )

Examples:

$f = x_1.x_2' + x_1'.x_2$

$= (x_1 + x_2).(x_1' + x_2')$

$h = x_1 + x_2.x_3$

$= (x_1'.(x_2' + x_3'))'$

Notation: Often the "." for AND is replaced by concatenation

e.g., $x_1x_2 + x_3$

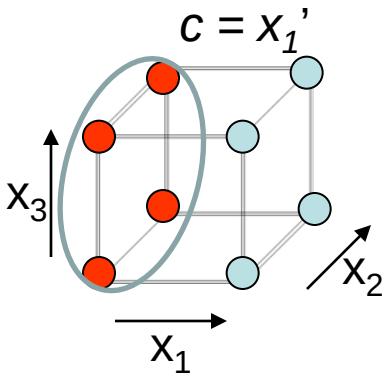
Question

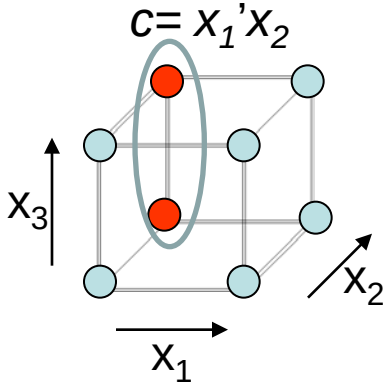
- How many Boolean formulae can be constructed with n variables?

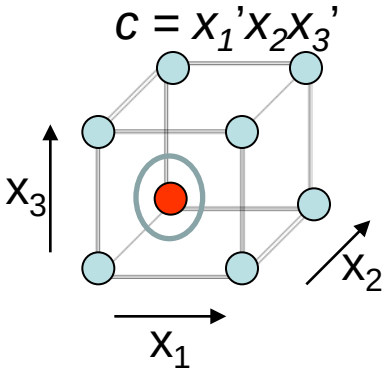
- How does this compare with the number of unique Boolean functions in n variables?

Cubes

- A cube (a.k.a. product term) is the conjunction (AND) of a set of literals
  - Also, a collection of vertices that forms a hypercube of lower dimension
- If $C \subseteq B^n$, and C has k literals, then $|C|$ covers 2^{n-k} vertices
- In an n-dimensional Boolean space B^n, a cube with n literals is called a minterm
- If a cube C $\subseteq f^1$ (f is a Boolean function), then C is an implicant of f





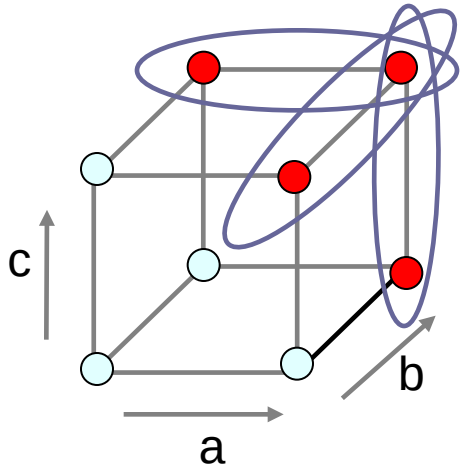


Sum of Products

Sum of Products (SOP)

- Any Boolean function can be represented by a sum of products

$$f = ab + ac + bc$$



- Can also be thought of as a set of cubes

$$F = \{ab, ac, bc\}$$

- A set of cubes that correctly represents f is called a ***cover*** of *f*.
- A function may have several different SOP representations or covers
- Example:

  $F_1 = \{ab, ac, bc\}$ and

  $F_2 = \{abc, a'bc, ab'c, abc'\}$

  are possible covers of the Boolean function

$$f = ab + ac + bc$$

Properties of a cover

- Each on-set vertex should be covered by at least one cube.
- No cube should cover any off-set vertex.

SOP : Minterm Canonical Form

- A Sum of Products representation for a function where each product is a minterm
  - For a function of n variables, each product has n literals representing all n variables

Example:

a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Minterm canonical form:

Product of Sums

- Product (conjunction) of terms, each of which is a sum (disjunction) of literals
  - E.g., f = (a + b + c)(a + b + c')(a' + b + c')(a' + b' + c)
- One-to-one transformation from SOP representation for f to POS representation for f' (complement of f)
  - Follows from De Morgan's law
- From truth table, use off-set to construct POS representation

Example:

a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

POS representation:

Binary Decision Tree

- Represent the function as a decision tree
- At each node, pick a variable and branch based on it's value
- Leaves of the tree contain constants (0,1)

Example:

a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Binary Decision Tree:

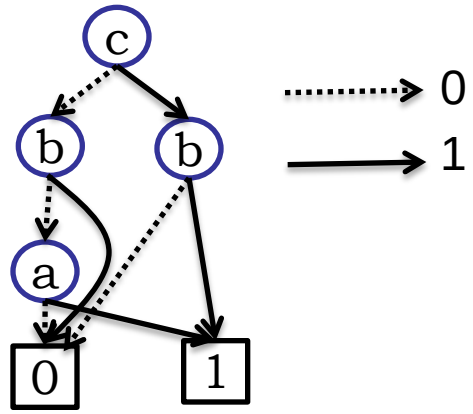
Binary Decision Diagram (BDD)

- Binary Decision Tree has large number of nodes
- Key idea: Share sub-trees and eliminate redundancy to reduce size
- More about BDDs later in the class

Example:

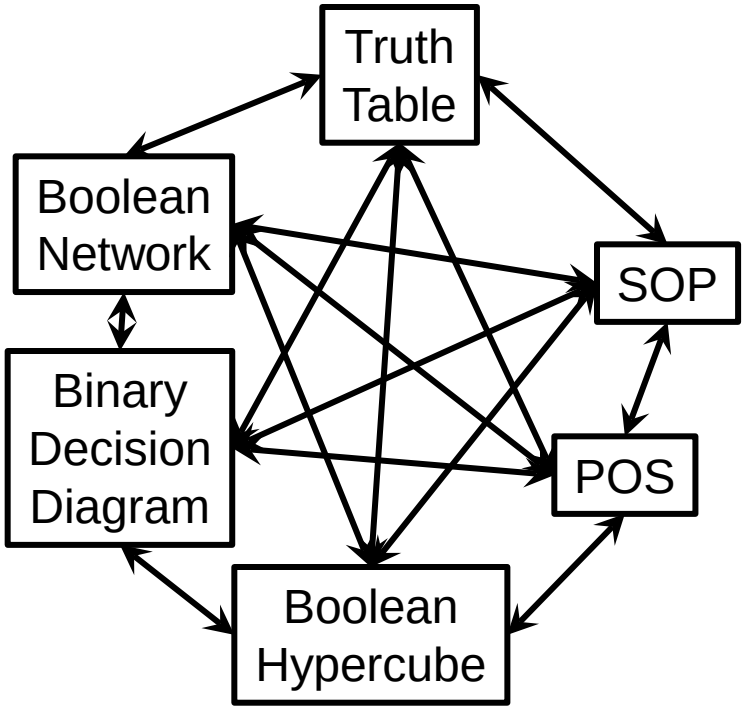
a b c	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Binary Decision Diagram:



Converting Between Boolean Function Representations

- All of the previously described representations are functionally equivalent
- Vary in their complexity (size), and ease of performing various operations
- Simple algorithms exist to convert from one form to another



Conversion : Example #1

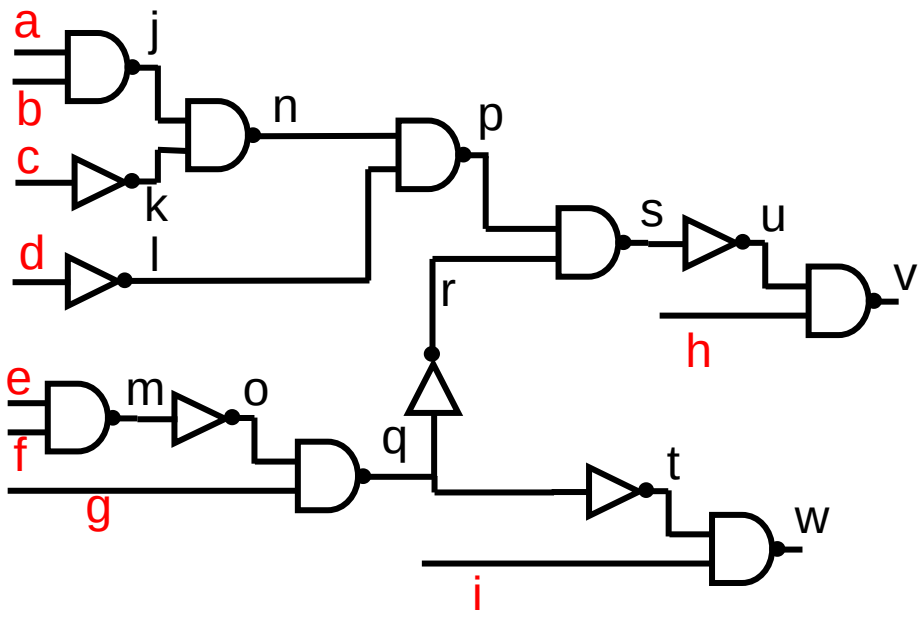
- How do you convert a general Boolean network (multi-level circuit) into SOP form?
  - Quick-and-dirty (exhaustive) algorithm

```
For each input vector (00…0 to 11…1) {
    Simulate the circuit;
    If (output == 1) {
        encode input vector as minterm;
    }
}
```

  - Works, but *guaranteed* to be exponential in the number of inputs
- There should be a better algorithm!

Conversion : Example #1

- j = (ab)' = a' + b'
- k = c'
- l = d'
- m = e' + f'
- n = j' + k' = (a'+b')' + (c')' = ab + c
- o = m' = ef
- p = n' + l' = (ab+c)' + d = a'c' + b'c' + d
- q = o'+g' = e' + f' + g'
- r = q' = efg
- s = p'+r' = (a+c)(b+c)d' + e' + f' + g' = abd' + cd' + e' + f' + g'
- t = q' = efg
- u = s' = (a'+b'+d)(c'+d)efg = a'c'efg + b'c'efg + defg
- v = u' + h' = abd' + cd' + e' + f' + g' + h'
- w = t' + i' = e'+f'+g'+i'



Notice the similarity to circuit simulation? Only difference is, we are propagating Boolean expressions, not 0/1 values. This is called **Symbolic Simulation**

Conversion : Example #2

- How do you convert a general Boolean network (multi-level circuit) into a Boolean formula that is linear in the circuit size?
  - Size(formula) = O(M) where M = no. of gates in circuit
  - SOP may be exponential in the worst case
  - Hints
    - Use variables to represent intermediate signals in the circuit
    - Compose the formula using a 1 : 1 mapping from each gate in the circuit into a piece of the formula