Boolean Algebra
(4541.554 Introduction to Computer-Aided Design)

School of EECS

Seoul National University

Boolean Algebra

- **Named after George Boole**
- **Given**
  - K **: a set of elements, e.g. {a,b,c,...}, $\{0,1\}^n$**
  - Φ **: a set of operations that act on members of K, e.g. {and,or}, {·,+}**
- **Huntington's postulate**
  - **closure: a** $\in \mathrm{K}$ **and b** $\in \mathrm{K} \Rightarrow$ **a+b** $\in \mathrm{K}$ **and ab** $\in \mathrm{K}$
  - **zero axiom: 0** $\in \mathrm{K}$ **such that a+0=a**

    unit axiom: 1 $\in \mathrm{K}$ **such that a1=a**
  - **commutativity: a+b=b+a and ab=ba**
  - **distributivity: a+bc=(a+b)(a+c) and a(b+c)=ab+ac**
  - **inverse axioms: a** $\in \mathrm{K} \Rightarrow$ **a'** $\in \mathrm{K}$ **such that a+a'=1 and aa'=0**
  - $|\mathrm{K}| \geq 2$

- **Theorems**
  - **0 is unique, 1 is unique**
  - **a+a=a, aa=a**
  - **a+1=1, a0=0**
  - **a+ab=a, a(a+b)=a**
  - **a’ is unique**
  - **(a+b)+c=a+(b+c), (ab)c=a(bc)**
  - **(a+b)’=a’b’, (ab)’=a’+b’**
  - **(a’)’=a**
  - **...**
- **Proof of a+1=1**

a+1 = (a+1)1	**unit axiom**
= (a+1)(a+a’)	**inverse axiom**
= a+1a’	**distributivity**
= a+a’	**commutativity + unit axiom**
= 1	**inverse axiom**

Boolean Functions

- **Representation**
  - **We consider only binary: B = {0, 1}**
  - **Boolean space: $B^n = \{0, 1\}^n$**
  - **Completely specified Boolean function**
    - **n inputs, m outputs**
      f: B^n --> B^m
  - **Incompletely specified Boolean function**
    - **f: B^n --> $\{0, 1, *\}^m$, * denotes don't care**

- Equation
  - Two-level: sum of products, product of sums

    x = ad+bd+a'c

    y = ac+bc
  - Multiple-level: factored form

    x = (a+b)d+a'c

    y = (a+b)c
- Tabular form
  - Columns: I/O

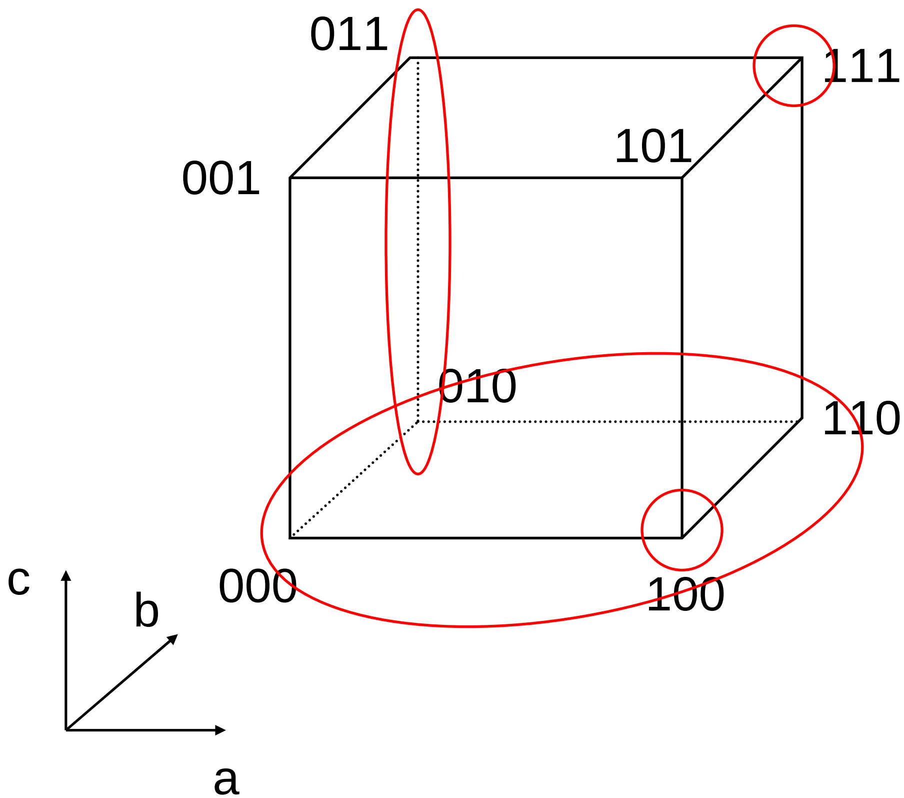
    Rows: Product terms

    Symbols: 1, 0, * (don't care),

    - (no information)
  - Truth table
    - no input don't cares
    - product term --> minterm
    - size = $2^{\#inputs}$

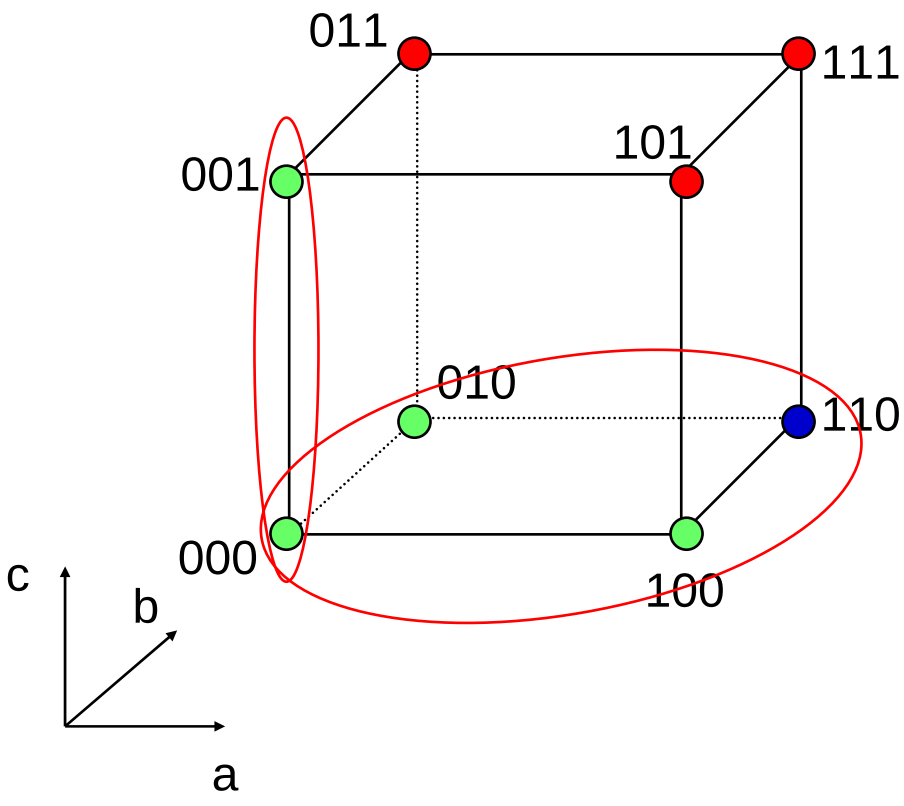
A	B	C	D	X	Y
1	*	*	1	1	-
*	1	*	1	1	-
0	*	1	*	1	-
1	*	1	*	-	1
*	1	1	*	-	1

- **Boolean n-cube**
  - **Boolean space: n-hypercube**
  - **Minterm: vertex**
  - **Product term: cube (k-dimensional cube)**



abc	111
ab'c'	100
a'b	01*
c'	**0

--> b + c'

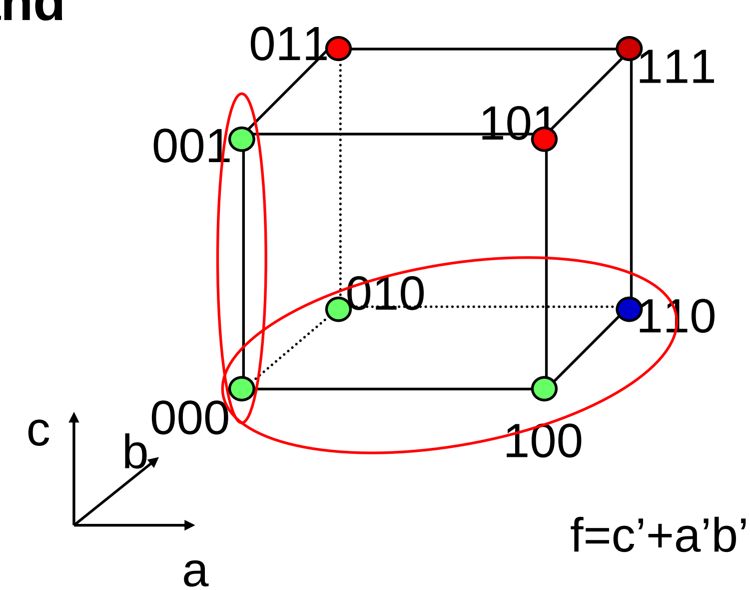


000	1	on-set
100	1	
010	1	
001	1	
101	0	off-set
011	0	
111	0	
110	*	dc-set

- **Definitions**
  - **On-set $X_i^{ON} \subseteq B^n$: set of input values x such that $f_i(x) = 1$ (for i-th output)**
  - **Off-set $X_i^{OFF} \subseteq B^n$: set of input values x such that $f_i(x) = 0$**
  - **Don't-care-set $X_i^{DC} \subseteq B^n$: set of input values x such that $f_i(x) = *$**
  - **$X_i^{ON} \cup X_i^{OFF} \cup X_i^{DC} = B^n$**
  - **Cover C: set of cubes such that**

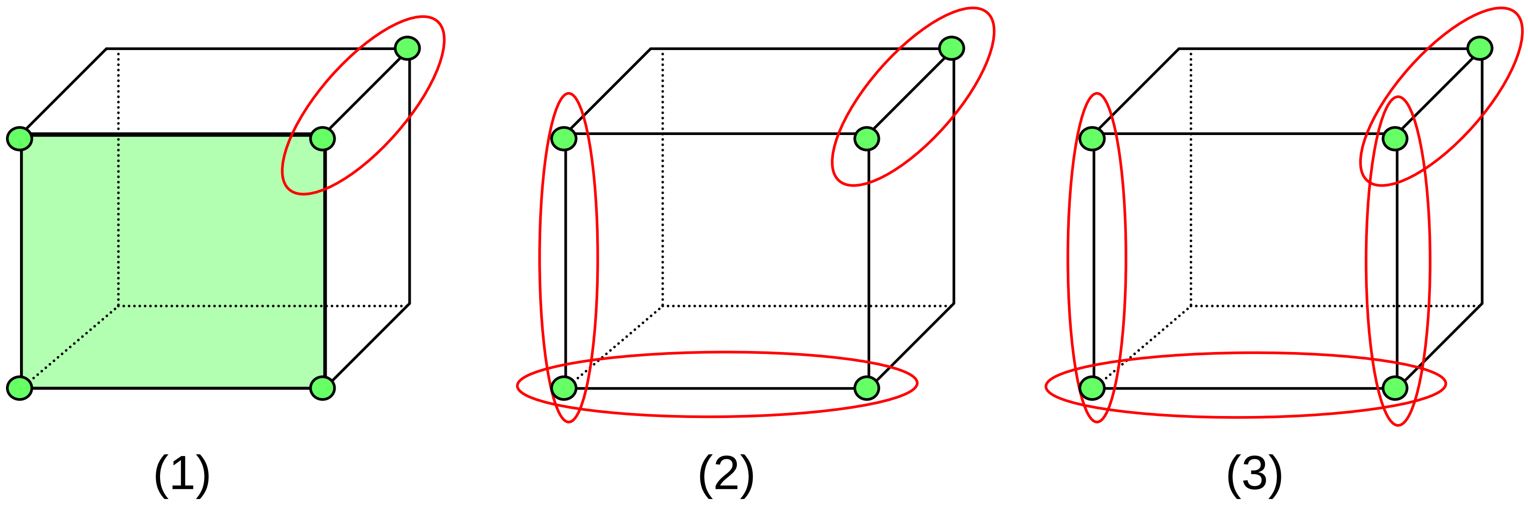
    $C_i \subseteq X_i^{ON} \cup X_i^{DC}$ and

    $C_i \supseteq X_i^{ON}$



000	1
100	1
010	1
001	1
101	0
011	0
111	*
110	*

- Cover cardinality |C| = Number of cubes in C
- Minimum cover (1)
  - Cover of minimum |C|
  - Usually +minimum number of literals
- Minimal (or irredundant) cover (2)
  - No proper subset of C is a cover
  - No cube in C is covered by the set of other cubes of C
- Minimal cover with respect to single cube containment (3)
  - $a \not\subseteq b$ for any distinct cubes $a, b \in C$

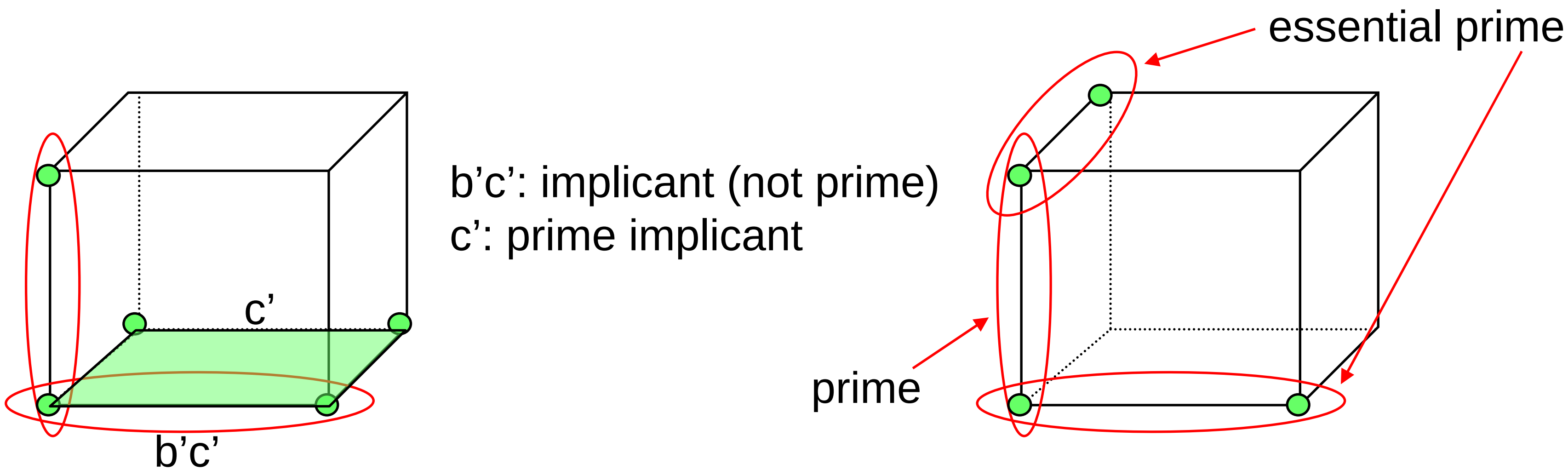


- **Implicant p_i: a cube that satisfies**

$$p_i \cap X_i^{OFF} = \varnothing$$

- **Prime implicant (prime cube, or prime): an implicant that is not contained in any other implicant**
- **Prime cover: a cover whose cubes are all prime implicant**
- **Essential prime: a prime that contains a minterm that is not contained in any other prime**
- **Distance between two cubes is equal to the number of conflicts in cube entries**

distance(11*, 100) = 1



- **Support of a Boolean function $f(x_1, x_2, ..., x_n)$ is the set $\{x_1, x_2, ..., x_n\}$**
- **Cofactor**
  - **Cofactor of $f(x_1, x_2, ..., x_i, ..., x_n)$ with respect to variable x_i is**

    $$f_{xi} = f(x_1, x_2, ..., 1, ..., x_n)$$

  - **Cofactor of $f(x_1, x_2, ..., x_i, ..., x_n)$ with respect to variable x_i' is**

    $$f_{x_i'} = f(x_1, x_2, ..., 0, ..., x_n)$$

  - **The number of variables/product terms can be reduced**

Boole's Expansion (Shannon's Expansion)

- $f = x_i f_{xi} + x_i' f_{xi'} = (x_i + f_{xi'}) (x_i' + f_{xi})$

  proof:
  - If f = constant --> $f_{xi} = f_{xi'} = f$ --> $x_i f_{xi} + x_i' f_{xi'} = (x_i + x_i') f = f$
  - If $f = x_i$ --> $f_{xi} = 1$, $f_{xi'} = 0$ --> $x_i f_{xi} + x_i' f_{xi'} = x_i = f$
  - If $f = x_j$, $i \neq j$ --> $f_{xi} = x_j$, $f_{xi'} = x_j$ --> $x_i f_{xi} + x_i' f_{xi'} = x_j = f$
  - Suppose $g = x_i g_{xi} + x_i' g_{xi'}$ and $h = x_i h_{xi} + x_i' h_{xi'}$
  - If f = g + h --> $f = g + h = (x_i g_{xi} + x_i' g_{xi'}) + (x_i h_{xi} + x_i' h_{xi'})$

    $= x_i (g_{xi} + h_{xi}) + x_i' (g_{xi'} + h_{xi'}) = x_i f_{xi} + x_i' f_{xi'}$
  - If f = g h --> $f = g h = (x_i g_{xi} + x_i' g_{xi'}) (x_i h_{xi} + x_i' h_{xi'})$

    $= x_i (g_{xi} h_{xi}) + x_i' (g_{xi'} h_{xi'}) = x_i (g h)_{xi} + x_i' (g h)_{xi'}$

    $= x_i f_{xi} + x_i' f_{xi'}$
  - If f = g' --> $f = g' = (x_i g_{xi} + x_i' g_{xi'})' = (x_i' + g_{xi}') (x_i + g_{xi'}')$

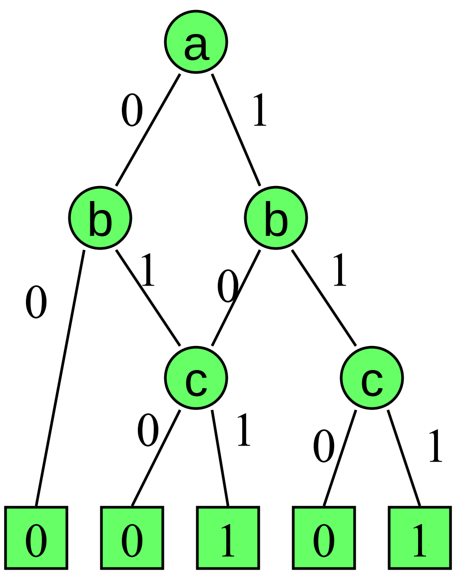
    $= x_i g_{xi}' + x_i' g_{xi'}' + g_{xi}' g_{xi'}' = x_i g_{xi}' + x_i' g_{xi'}'$

    $= x_i (g')_{xi} + x_i' (g')_{xi'} = x_i f_{xi} + x_i' f_{xi'}$

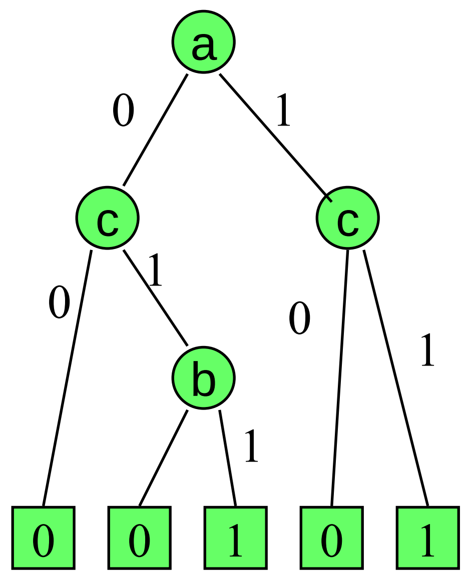
- $f = (x_i + f_{xi'}) (x_i' + f_{xi})$

Binary Decision Diagrams

- **Ordered binary decision diagram (OBDD)**
  - **Variables are ordered.**
  - **Can be transformed to a canonical form (ROBDD) --> uniquely characterize the function**

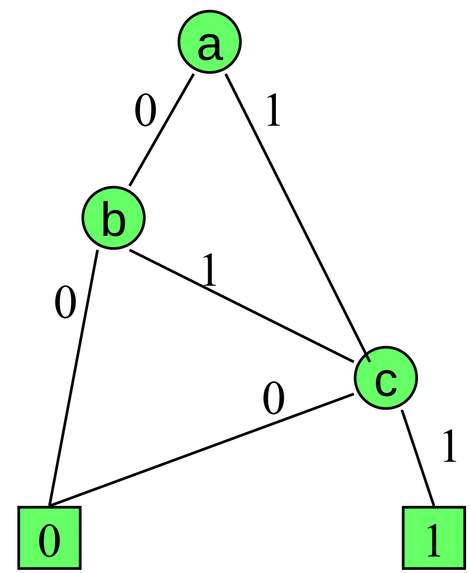


variable order (a,b,c)



variable order (a,c,b)

f=(a+b)c



variable order (a,b,c)

ROBDD

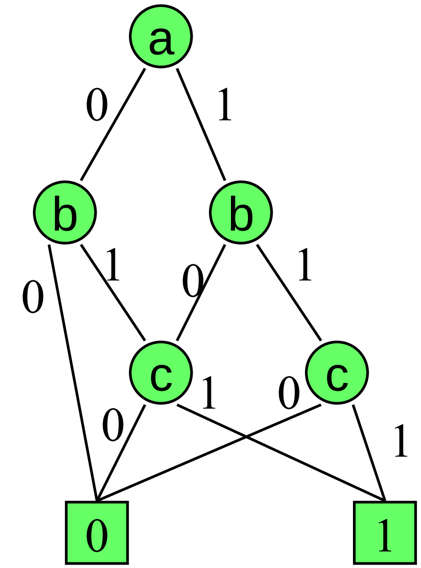
- **f^v: function represented by vertex v**
- **If v is a leaf with value(v)=1, then $f^v = 1$**

  If v is a leaf with value(v)=0, then $f^v = 0$

  If v is not a leaf, then

  $f^v = x_i' f^{low(v)} + x_i f^{high(v)}$

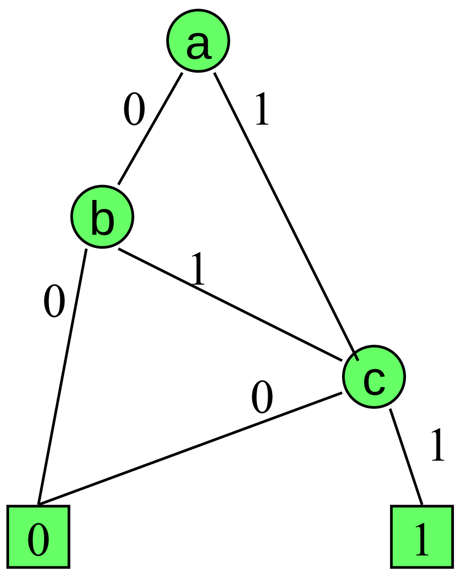
  where i = index(v)



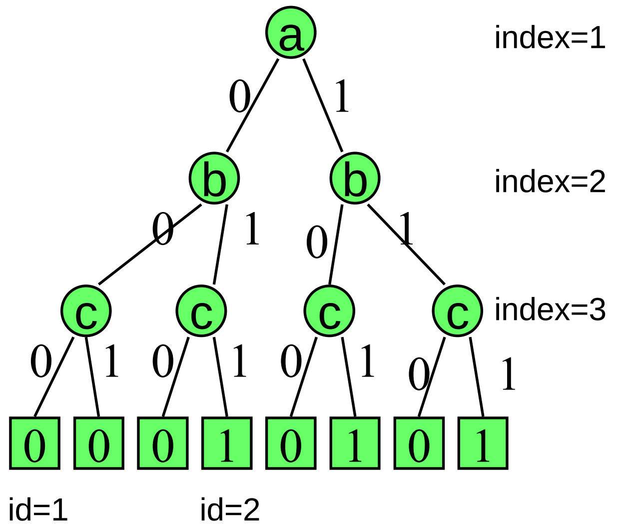
- **ROBDD**
  - **low(v) ≠ high(v) for any vertex v**
    - **id(low(v)) = id(high(v))**

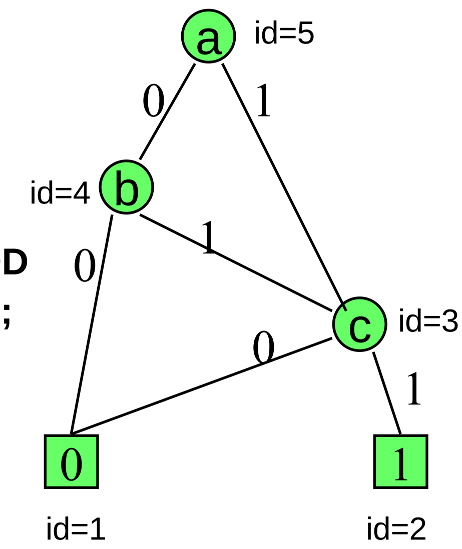
      --> v is redundant (remove)
  - **Subgraphs rooted in u and v are not isomorphic for any pair {u, v}**
    - **index(u) = index(v),**
      id(low(u)) = id(low(v)), and
      id(high(u)) = id(high(v))

      --> id(u) = id(v)

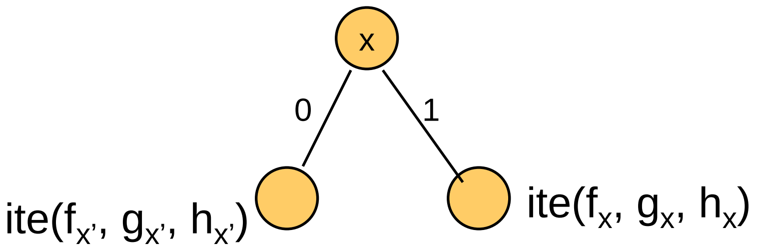


```
REDUCE(OBDD) {
    set id(v) = 1 for all leaves v with value(v) = 0;
    set id(v) = 2 for all leaves v with value(v) = 1;
    initialize ROBDD with two leaves with id = 1 and id = 2 respectively;
    nextid = 2;
    for (i = n to 1 with i = i - 1){
        V(i) = {v | index(v) = i};
        foreach (v ∈ V(i)) {
            if (id(low(v)) = id(high(v))) {
                id(v) = id(low(v));
                drop v from V(i);
            }
            else key(v) = {id(low(v)), id(high(v))};
        }
        oldkey = {0, 0}
        foreach v ∈ V(i) sorted by key(v) {
            if (key(v) = oldkey) id(v) = nextid;
            else {
                nextid = nextid + 1;
                id(v) = nextid;
                oldkey = key(v);
                add v to ROBDD with edges to vertices in ROBDD
                    whose ids equal those of low(v) and high(v);
            }
        }
    }
}
```



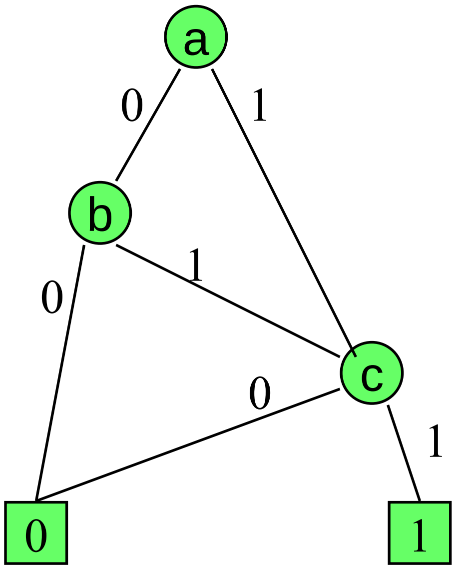


- **Number of vertices in OBDD is exponential**
- **Construct ROBDD without generating OBDD**
  - **f g = ite (f, g, 0)**
    f + g = ite (f, 1, g)
    f' = ite (f, 0, 1)
    ---> reduce the problem
  - **z = ite(f, g, h) = f g + f' h**
    $z = xz_x + x'z_{x'}$
    $= x\,(f\,g + f'\,h)_x + x'\,(f\,g + f'\,h)_{x'}$
    $= x\,(f_x\,g_x + f'_x\,h_x) + x'\,(f_{x'}\,g_{x'} + f'_{x'}\,h_{x'})$
    $= ite(x,\ ite(f_x, g_x, h_x), ite(f_{x'}, g_{x'}, h_{x'}))$
    ---> $ite(f, g, h) = ite(x,\ ite(f_x, g_x, h_x), ite(f_{x'}, g_{x'}, h_{x'}))$
    ---> recursion
    terminal cases:
    - **ite(f, 1, 0) = f**
    - **ite(f, 0, 1) = f'**
    - **ite(1, g, h) = g**
    - **ite(0, g, h) = h**
    - **ite(f, g, g) = g**



– **example**

- **ite(f, g, h) = ite(x, ite(f_x, g_x, h_x), ite($f_{x'}$, $g_{x'}$, $h_{x'}$))**
- **ac + bc = ite(ac, 1, bc) = ite(f, g, h)**
  = ite(a, ite(c, 1, bc), ite(0, 1, bc))
  = ite(a, ite(b, ite(c, 1, c), ite(c, 1, 0)), bc)
  = ite(a, ite(b, c, c), ite(b, c, 0))
  = ite(a, c, ite(b, ite(1, c, 0), ite(0, c, 0)))
  = ite(a, c, ite(b, c, 0))



```
ITE (f, g, h) {
   if (terminal case) {
         r = result;
         if r is not trivial
            transform r to ite( ) form and call ITE and return the result;
         else return (r);
 }
   else {
         if (computed table has entry {(f, g, h), r})
             return (r from computed table);
         else {
             x = top variable of f, g, h;
             t = ITE(f_x, g_x, h_x);
             e = ITE(f_x', g_x', h_x');
           if (t == e) return (t);
           r = find_or_add_unique_table(x, t, e);
           update computed table with {(f, g, h), r};
           return (r);
    }
 }
}
```