COE 561
Digital System Design & Synthesis
Logic Synthesis Background

Dr. Aiman H. El-Maleh

Computer Engineering Department

King Fahd University of Petroleum & Minerals

[Adapted from slides of Prof. G. De Micheli: Synthesis & Optimization of Digital Circuits]

Outline

- Boolean Algebra
- Boolean Functions
- Basic Definitions
- Representations of Boolean Functions
- Binary Decision Diagrams (BDDs)
  - Ordered BDDs (OBDDs)
  - Reduced Ordered BDDs (ROBDDs)
- If-then-else (ITE) DAGS
- Satisfiability and Minimum Cover Problems
- Branch and Bound Algorithm

Boolean Algebra

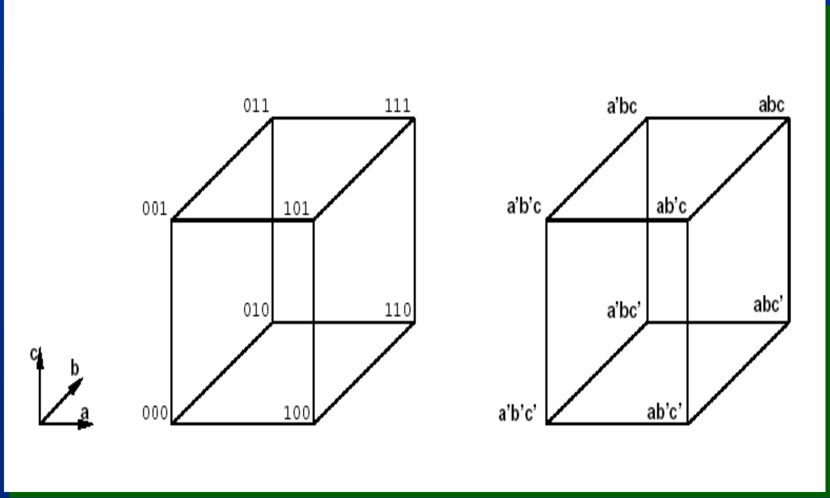
- Boolean algebra
  - Quintuple (B,+, . , 0, 1)
  - Satisfies *commutative* and *distributive* laws
  - Identity elements are 0 and 1.
  - Each element has a complement: $a + a'=1$; $a . a' = 0$
  - Binary Boolean algebra B = {0, 1}
- Some properties of Boolean algebraic systems

Associativity	**a+(b+c)=(a+b)+c**	**a(bc)=(ab)c**
Idempotence	**a+a=a**	**a.a=a**
Absorption	**a+(ab)=a**	**a(a+b)=a**
De Morgan	**(a+b)'=a'.b'**	**(a.b)'=a'+b'**
Involution	**(a')'=a**	

Boolean Functions

- Boolean function
  - **Single output:** $f : B^n \rightarrow B$
  - **Multiple output:** $f : B^n \rightarrow B^m$
  - **Incompletely specified**
    - **don't care symbol *.**
    - $f : B^n \rightarrow \{0,1,*\}^m$

3-dimensional Boolean Space



- **Don't care conditions**
  - We don't care about the value of the function.
  - Related to the environment:
    - Input patterns that never occur.
    - Input patterns such that some output is never observed.
  - Very important for synthesis and optimization.

Definitions …

- **Scalar function**
  - ON-Set: subset of the domain such that *f* is true.
  - Off-Set: subset of the domain such that *f* is false.
  - Don’t care Set: subset of the domain such that *f* is a don't care.
- **Multiple-output function**
  - Defined for each component.
- **Boolean literal: variable or its complement.**
- **Product or cube: product of literals.**
- **Implicant: product implying a value of a function (usually TRUE).**
  - Hypercube in the Boolean space.
- **Minterm: product of all input variables implying a value of a function (usually TRUE).**
  - Vertex in the Boolean space.

… Definitions …

- Let $f(x_1,x_2,\dots,x_n)$ be a Boolean function of n variables.
- The set $(x_1,x_2,\dots,x_n)$ is called the *support* of the function.
- The *cofactor* of $f(x_1,x_2,\dots,x_i,\dots,x_n)$ with respect to variable x_i is $f_{x_i} = f(x_1,x_2,\dots,x_i=1,\dots,x_n)$
- The *cofactor* of $f(x_1,x_2,\dots,x_i,\dots,x_n)$ with respect to variable x_i' is $f_{x_i'} = f(x_1,x_2,\dots,x_i=0,\dots,x_n)$
- Theorem: Shannon's Expansion

$$Let\ f : B^n \rightarrow B. Then\quad f(x_1,x_2,\dots,x_i,\dots,x_n) = x_i . f_{x_i} + x_i' . f_{x_i'}$$
$$= (x_i + f_{x_i'}).(x_i' + f_{x_i})\ \forall \mathrm{i} = 1,2,\dots,\mathrm{n}$$

- Any function can be expressed as *sum of products* (*product of sums*) of n literals, minterms (maxterms), by recursive expansion.

… Definitions …

- **Example: $f = ab + ac + bc$**
  - $f_a = b + c$
  - $f_{a'} = bc$
  - $F = a\, f_a + a'\, f_{a'} = a\,(b + c) + a'\,(bc)$
- **A Boolean function can be interpreted as the set of its minterms.**
- **Operations and relations on Boolean functions can be viewed as operations on their minterm sets**
  - Sum of two functions is the Union ($\cup$) of their minterm sets
  - Product of two functions is the Intersection ($\cap$) of their minterm sets
  - Implication between two functions corresponds to containment ($\subseteq$) of their minterm sets
    - $f_1 \rightarrow f_2 \equiv f_1 \subseteq f_2 \equiv f_1' + f_2 = 1$

… Definitions …

- A function $f(x_1,x_2,\ldots,x_i,\ldots,x_n)$ is **positive (negative) Unate** with respect to variable x_i if $f_{xi} \supseteq f_{xi'}$ ($f_{xi} \subseteq f_{xi'}$).
- A function is (positive/negative) **Unate** if it is (positive/negative) unate in all support variables, otherwise it is **Binate** (or mixed).
- **Example: f= a + b + c'**
  - f is positive unate with respect to variable a
    - $f_a=1 \supseteq f_{a'}= b + c'$
    - Minterms of f_a ={bc, b'c,bc',b'c'} $\supseteq$ minterms of $f_{a'}$={bc,bc',b'c'}
  - f is positive unate with respect to variable b
  - f is negative unate with respect to variable c
  - Thus, f is binate.

… Definitions

- The **Boolean Difference** of a function $f(x_1,x_2,\ldots,x_i,\ldots,x_n)$ with respect to variable x_i is $\partial f / \partial x_i = f_{xi} \oplus f_{xi'}$
  - Indicates whether f is sensitive to changes in x_i
- The **Consensus** of a function $f(x_1,x_2,\ldots,x_i,\ldots,x_n)$ with respect to variable x_i is $f_{xi} \,.\, f_{xi'}$
  - Represents the component that is independent of x_i
- The **Smoothing** of a function $f(x_1,x_2,\ldots,x_i,\ldots,x_n)$ with respect to variable x_i is $f_{xi} + f_{xi'}$
  - Corresponds to dropping the variable from the function
- **Example: f= ab + ac + bc**
  - f_a=b+c $\quad$ $f_{a'}$= bc
  - Boolean difference = $f_a \oplus f_{a'}$= (b+c) $\oplus$ bc = b'c+bc'
  - Consensus = $f_a \,.\, f_{a'}$= (b+c) . bc = bc
  - Smoothing = $f_a + f_{a'}$= (b+c) + bc = b+c

Boolean Expansion Based on Orthonormal Basis …

- Let ϕ_i , i=1,2, …,k be a set of Boolean functions such that $\Sigma_{i=1 \text{ to } k}\ \phi_i = 1$ and $\phi_i \cdot \phi_j = 0$ for $\forall\ i \neq j \in \{1,2,\ldots,k\}$.
- An Orthonormal Expansion of a function f is

  $f = \Sigma_{i=1 \text{ to } k}\ f_{\phi i} \cdot \phi_i$
- $f_{\phi i}$ is called the generalized cofactor of f w.r.t. $\phi_i\ \forall$ i.
- The generalized cofactor may not be unique
  - $f \cdot \phi_i \subseteq f_{\phi i} \subseteq f + \phi_i'$
- Example: $f = ab+ac+bc$; $\phi_1 = ab$; $\phi_2 = a'+b'$;
  - $ab \subseteq f_{\phi 1} \subseteq 1$; let $f_{\phi 1} = 1$
  - $a'bc+ab'c \subseteq f_{\phi 2} \subseteq ab+bc+ac$; let $f_{\phi 2} = a'bc+ab'c$
  - $f = \phi_l\ f_{\phi l} + \phi_2\ f_{\phi 2} = ab\,(1) + (a'+b')(a'bc+ab'c) = ab+bc+ac$

… Boolean Expansion Based on Orthonormal Basis …

- **Theorem**
  - Let f, g, be two Boolean functions expanded with the same orthonormal basis ϕ_i , i=1,2, …,k
  - Let $\otimes$ be a binary operator on two Boolean functions

$$f \otimes g = \sum_{i=1}^{k} \Phi_i.(f_{\Phi_i} \otimes g_{\Phi_i})$$

- **Corollary**
  - Let f, g, be two Boolean functions with support variables {x_i, i=1,2, …,n}.
  - Let $\otimes$ be a binary operator on two Boolean functions

$$f \otimes g = x_i.(f_{x_i} \otimes g_{x_i}) + x_i'.(f_{x_i'} \otimes g_{x_i'})$$

… Boolean Expansion Based on Orthonormal Basis

- **Example:**
  - Let $f = ab + c$; $g = a'c + b$; Compute $f \oplus g$
  - Let $\phi_1 = a'b'$; $\phi_2 = a'b$; $\phi_3 = ab'$; $\phi_4 = ab$;
  - $f_{\phi 1} = c$; $f_{\phi 2} = c$; $f_{\phi 3} = c$; $f_{\phi 4} = 1$;
  - $g_{\phi 1} = c$; $g_{\phi 2} = 1$; $g_{\phi 3} = 0$; $g_{\phi 4} = 1$;
  - $f = a'b' (c \oplus c) + a'b (c \oplus 1) + ab' (c \oplus 0) + ab (1 \oplus 1)$
    $= a'bc' + ab'c$
  - $F = (ab+c) \oplus (a'c+b) = (ab+c)(a+c')b' + (a'+b')c'(a'c+b)$
    $= (ab+ac)b' + (a'c+a'b)c' = ab'c + a'bc'$

Representations of Boolean Functions

- **There are three different ways of representing Boolean functions:**
  - Tabular forms
    - Personality matrix
    - Truth table
    - Implicant table
  - Logic expressions
    - Expressions of literals linked by the + and . Operators
    - Expressions can be nested by parenthesis
    - Two-level: sum of products or products of sum
    - Multilevel: factored form
  - Binary decisions diagrams
    - Represents a set of binary-valued decisions, culminating in an overall decision that can be either TRUE or FALSE

Tabular Representations

- **Truth table**
  - List of all minterms of a function.
- **Implicant table or cover**
  - List of implicants of a function sufficient to define a function.
- **Implicant tables are smaller in size.**
- **Example: x = ab+a'c; y = ab+bc+ac**

Truth Table

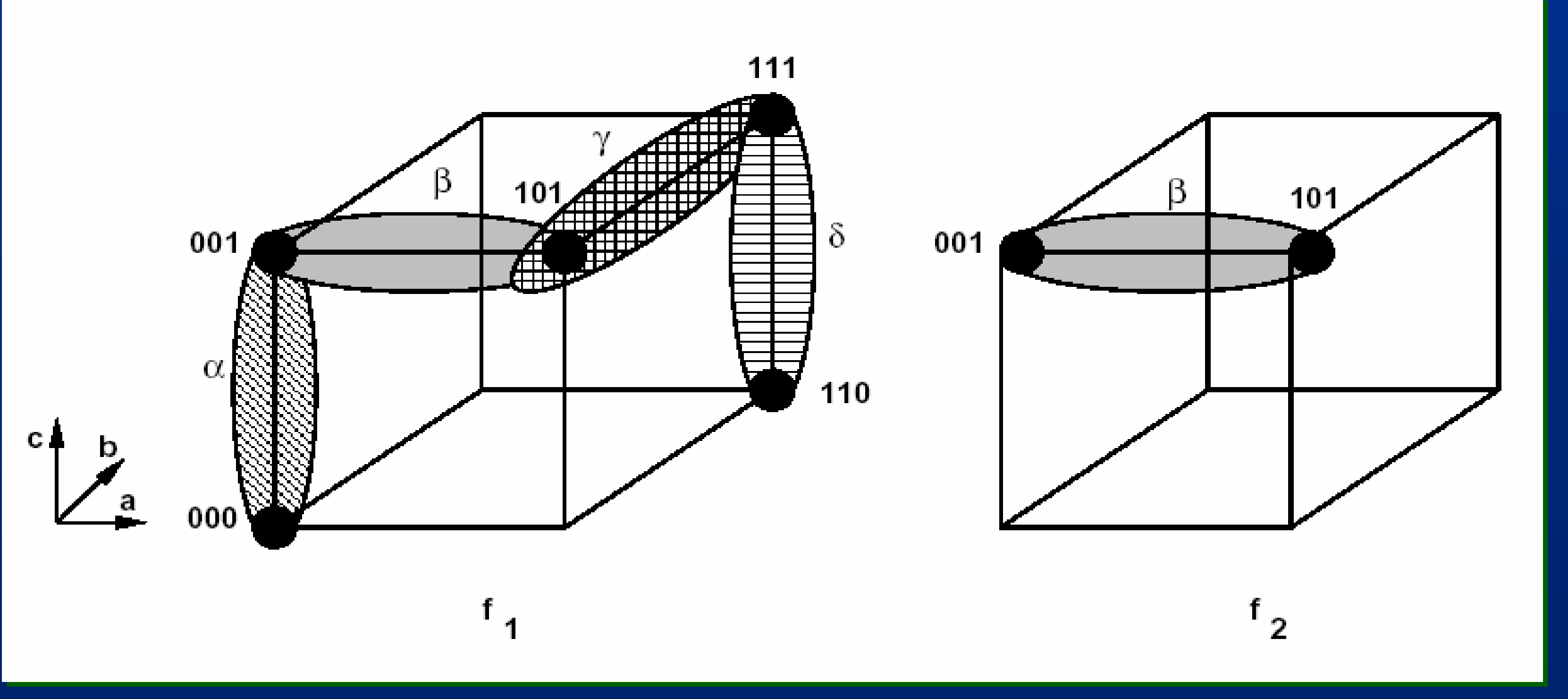
abc	xy
000	00
001	10
010	00
011	11
100	00
101	01
110	11
111	11

Implicant Table

abc	xy
001	10
*11	11
101	01
11*	11

Cubical Representation of Minterms and Implicants

- f1 = a'b'c'+a'b'c+ab'c+abc+abc'= a'b'+b'c+ac+ab
- f2 = a'b'c+ab'c = b'c



Binary Decision Diagrams …

- **Binary decision diagrams (BDDs) can be represented by trees or rooted DAGs, where decisions are associated with vertices.**
- **Ordered binary decision diagrams (OBDDs) assume an ordering on the decision variables.**
  - Can be transformed into canonical forms, reduced ordered binary decision diagrams (ROBDDs)
  - Operations on ROBDDs can be made in polynomial time of their size i.e. vertex set cardinality
  - Size of ROBDDs depends on **ordering of variables**
    - Adder functions are very sensitive to variable ordering
      - Exponential size in worst case
      - Linear size in best case
    - Arithmetic multiplication has exponential size regardless of variable order.

… Binary Decision Diagrams …

- **An OBDD is a rooted DAG with vertex set V. Each non-leaf vertex has as attributes**
  - a pointer index(v) $\in$ {1,2,…n} to an input variable $\{x_1, x_2, \ldots, x_i, \ldots, x_n\}$.
  - Two children low(v) and high(v) $\in$ V.
- **A leaf vertex v has as an attribute a value value(v) $\in$ B.**
- **For any vertex pair {v,low(v)} (and {v,high(v)}) such that no vertex is a leaf, index(v)<index(low(v)) (index(v)<index(high(v))**
- **An OBDD with root v denotes a function f^v such that**
  - 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 and index(v)=i, then $f^v= x_i' \cdot f^{low(v)} + x_i \cdot f^{high(v)}$

… Binary Decision Diagrams

- **Example: f=(a+b)c**
  - Vertices {v1,v2,v3,v4,v5} (Fig. 2.20 (c))
  - Variable x_1=a, x_2=b, x_3=c;
  - v1 is the root; index(v1)=1 meaning that v1 is related to first variable in the order i.e. x_1=a

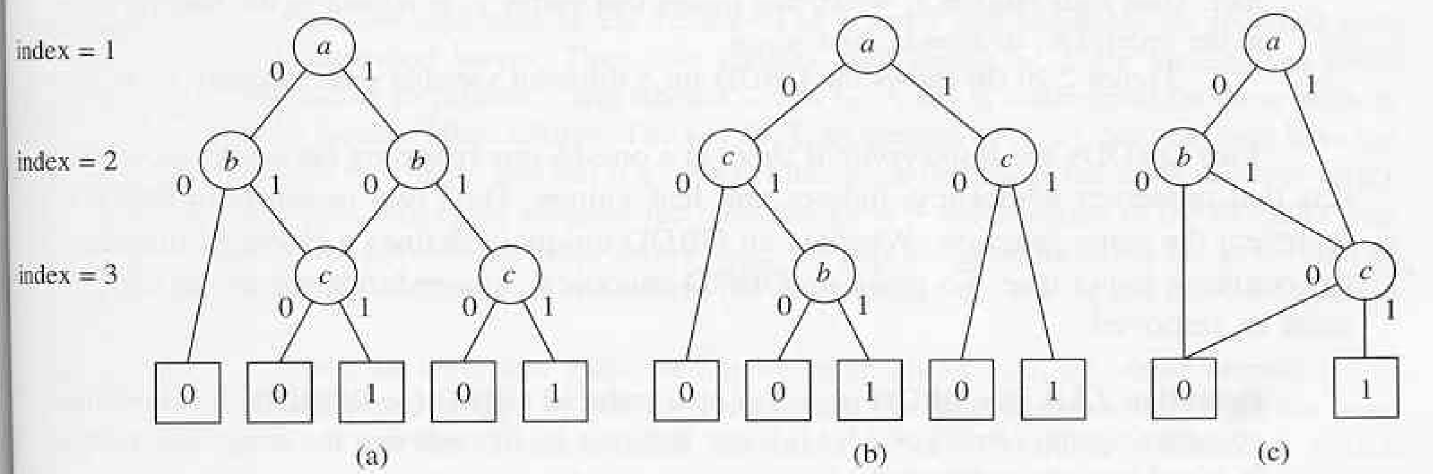


FIGURE 2.20
Binary decision diagrams for $f = (a + b)c$: (a) OBDD for the variable order (a, b, c). (b) OBDD for the variable order (a, c, b). (c) ROBDD for the variable order (a, b, c).

Reduced Binary Decision Diagrams …

- **Two OBDDs are isomorphic if there is a one-to-one mapping between the vertex set that preserves adjacency, indices and leaf values.**
- **Two isomorphic OBDDS represent the same function.**
- **An OBDD is said to be reduced OBDD (ROBDD) if**
  - It contains no vertex v with low(v)=high(v)
  - Not any pair {u,v} such that the subgraphs rooted in u and in v are isomorphic.
- **ROBDDs are canonical**
  - All equivalent functions will result in the same ROBDD.

… Reduced Binary Decision Diagrams …

```
REDUCE(OBDD){
    Set id(v) = 1 to all leaves v ∈ V with value(v) = 0;
    Set id(v) = 2 to all leaves v ∈ V with value(v) = 1;
    Initialize ROBDD with two leaves with id = 1 and id = 2 respectively;
    nextid = 2;                          /* nextid is the next available identifier value */
    for (i = n to 1 with i = i − 1){
        V(i) = {v ∈ V : index(v) = i};
        foreach (v ∈ V(i)){                      /* consider vertices at level i */
            if (id(low(v)) = id(high(v))){
                id(v) = id(low(v));                      /* redundant vertex */
                Drop v from V(i);
            }
            else
                key(v) = id(low(v)), id(high(v));
                                  /* define key(v) as the identifier pair of v's children */
        }
        oldkey = 0, 0;                   /* initial key that cannot be matched by any vertex */
        foreach v ∈ V(i) sorted by key(v) {
            if (key(v) = oldkey)                     /* graph rooted at v is redundant */
                id(v) = nextid
            else {                        /* nonredundant vertex to receive new identifier value */
                nextid = nextid + 1;
                id(v) = nextid;
                oldkey = key(v);
                Add v to ROBDD with edges to vertices in ROBDD
                    whose id equal those of low(v) and high(v);
            }
        }
    }
}
```

Reduced Binary Decision Diagrams …

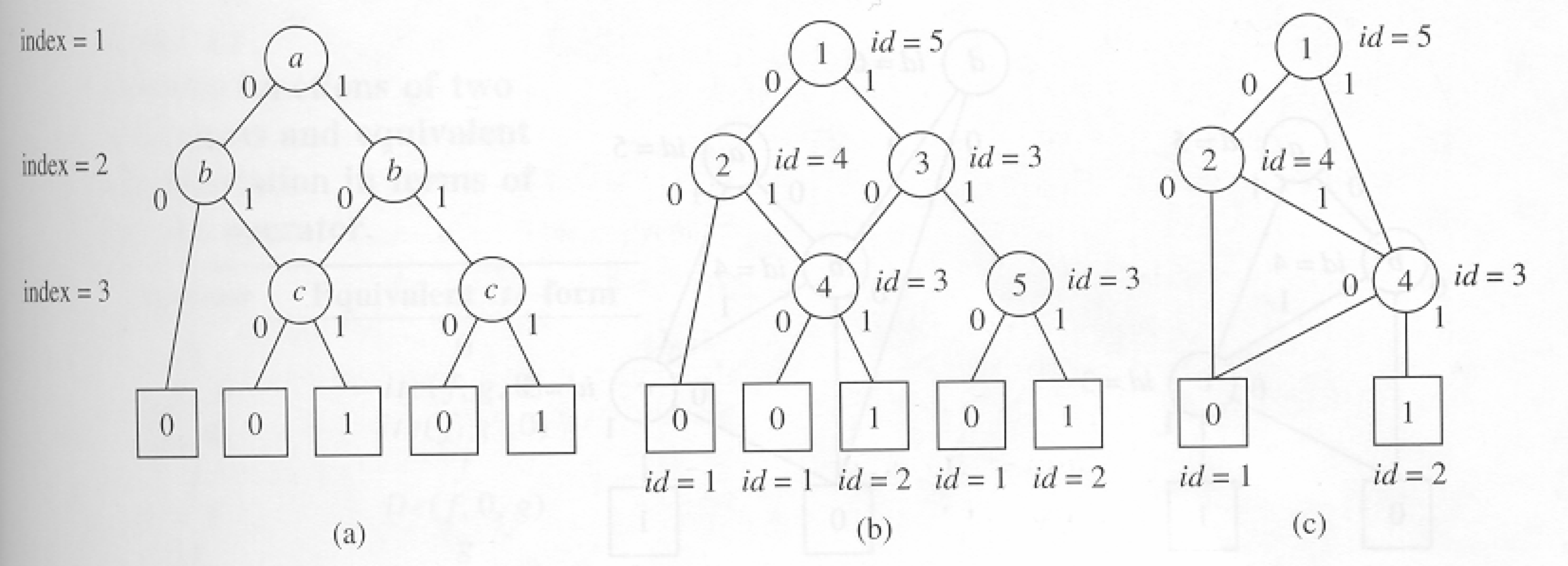


FIGURE 2.21
Binary decision diagrams for $f = (a + b)c$: (a) OBDD for the variable order (a, b, c). (b) OBDD with identifiers. (c) ROBDD for the variable order (a, b, c).

If-then-else (ITE) DAGs …

- **ROBDD construction and manipulation can be done with the *ite* operator.**
- **Given three scalar Boolean functions f, g and h**
  - $Ite(f, g, h) = f \cdot g + f' \cdot h$
- **Let z=ite(f, g, h) and let x be the top variable of functions f, g and h.**
- **The function z is associated with the vertex whose variable is x and whose children implement $ite(f_x,g_x,h_x)$ and $ite(f_{x'},g_{x'},h_{x'})$.**
  - $z = x\, z_x + x'\, z_{x'}$
  - $= x(f g + f' h)_x + x' (f g + f' h)_{x'}$
  - $= x(f_x\, g_x + f'_x\, h_x) + x' (f_{x'}\, g_{x'} + f'_{x'}\, h_{x'})$
  - $=ite(x, ite(f_x,g_x,h_x) , ite(f_{x'},g_{x'},h_{x'}))$

… If-then-else (ITE) DAGs

- **Terminal cases of *ite* operator**
  - Ite(f,1,0)=f, ite(1,g,h)=g, ite(0, g, h)=h, ite(f, g, g)=g and ite(f, 0, 1)=f'.
- **All Boolean functions of two arguments can be represented in terms of *ite* operator.**

Operator	Equivalent *ite* form
0	0
$f \cdot g$	$ite(f, g, 0)$
$f \cdot g'$	$ite(f, g', 0)$
f	f
$f'g$	$ite(f, 0, g)$
g	g
$f \oplus g$	$ite(f, g', g)$
$f + g$	$ite(f, 1, g)$
$(f + g)'$	$ite(f, 0, g')$
$f \overline{\oplus} g$	$ite(f, g, g')$
g'	$ite(g, 0, 1)$
$f + g'$	$ite(f, 1, g')$
f'	$ite(f, 0, 1)$
$f' + g$	$ite(f, g, 1)$
$(f \cdot g)'$	$ite(f, g', 1)$
1	1

ITE Algorithm …

```
ITE(f, g, h){
    If (terminal case)
        return (r = trivial result)
    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)
        }
    }
}
```

… ITE Algorithm

- **Uses two tables**
  - Unique table: stores ROBDD information in a strong canonical form
    - Equivalence check is just a test on the equality of the identifiers
    - Contains a key for a vertex of an ROBDD
    - Key is a triple of variable, identifiers of left and right children
  - Computed table: to improve the performance of the algorithm
    - Mapping between any triple (f, g, h) and vertex implementing ite(f, g, h).

Applications of ITE DAGs

- **Implication of two functions is Tautology**
  - $f \rightarrow g \equiv f' + g = 1$
  - Check if ite(f, g, 1) has identifier equal to that of leaf value 1
  - Alternatively, a function associated with a vertex is tautology if both of its children are tautology
- **Functional composition**
  - Replacing a variable by another expression
  - $f_{x=g} = f_x\, g + f_{x'}\, g' = ite(g, f_x, f_{x'})$
- **Consensus**
  - $f_x \,.\, f_{x'} \equiv ite(f_x, f_{x'}, 0)$
- **Smoothing**
  - $f_x + f_{x'} \equiv ite(f_x, 1, f_{x'})$

Satisfiability …

- Many synthesis and optimization problems can be reduced to a fundamental one: satisfiability.
- A Boolean function is satisfiable if there exists an assignment of Boolean values to the variables that makes the function TRUE.
- Most common formulation requires the function to be expressed in a product of sum form
  - Sum terms are called clauses
  - Assignment must make all clauses true
- Satisfiability problem is Intractable
  - 3-satisfiability (i.e. clauses with max. 3 literals) is intractable
  - 2-satisfiability can be solved in polynomial time

… Satisfiability

- **Example**
  - F=(a+b+c')(a+b'+c')(a+b'+c)(a'+b+c)(a'+b+c')(a'+b'+c')(a'+b'+c)
  - Find an input assignment that makes F=1
- **Solution**
  - A=1, B=1, C=0 => Fails
  - A=0, B=1, C=0 => Fails
  - A=1, B=0, C=1 => Fails
  - A=0, B=0, C=1 => Fails
  - A=1, B=1, C=1 => Fails
  - A=0, B=1, C=1 => Fails
  - A=1, B=0, C=0 => Fails
  - A=0, B=0, C=0 => Success!!

Satisfiability Formulation as Zero-One Linear Programming (ZOLP) Problem

- **Satisfiability problem can be modeled as a ZOLP**
- **Example: Satisfiability problem**
  - (a+b)(a'+b'+c)
  - Possible solution: a=1; b=1; c=1
- **ZOLP modeling**
  - $a + b \geq 1$
  - $(1-a)+(1-b)+c \geq 1$
  - $a, b, c \in B$
- **Minimum-cost satisfiability problem**
  - Find $x \in B^n$ that minimizes the cost $c^T x$ where c is a weight vector.

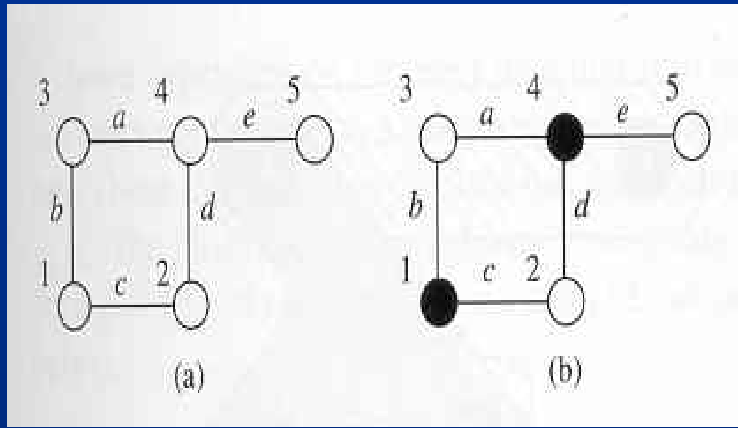
Minimum Covering Problem

- **Given a collection C (called groups) of subsets of a finite set S. A minimum-covering problem is the search of a minimum number of subsets from C that cover S.**
- **Let $A \in B^{nxm}$, where #rows=n=|S| and #columns=m=|C|**
  - A cover corresponds to a subset of columns having at least a 1 entry in all rows of A.
  - Corresponds to selecting $x \in B^m$, such that $Ax \geq 1$
  - Minimum-weighted cover corresponds to selecting $x \in B^m$, such that $Ax \geq 1$ and $c^T x$ is minimum.
- Intractable.
- Exact method
  - Branch and bound algorithm.
- Heuristic methods.

Minimum-Vertex Cover Example

Vertex/edge incidence matrix

$$A_I = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

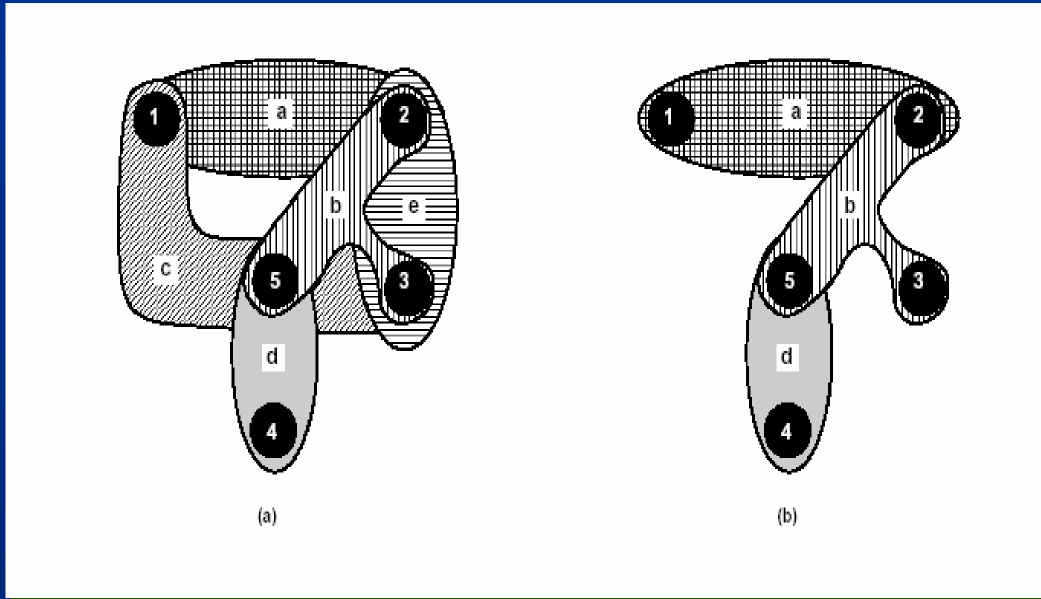


- **Minimum vertex cover**
  - Edge set corresponds to S and vertex set to C
  - $A = A_I^T$ and $c = 1$.
- Possible covers: $x^1=[10010]^T$, $x^2=[01101]^T$, $x^3=[01111]^T$
- Note that $Ax \geq 1$ for $x = x^1, x^2, x^3$
- Vector x^1 is a minimum cover

Minimum-Edge Cover Example

Vertex/edge incidence matrix

$$A_I = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$



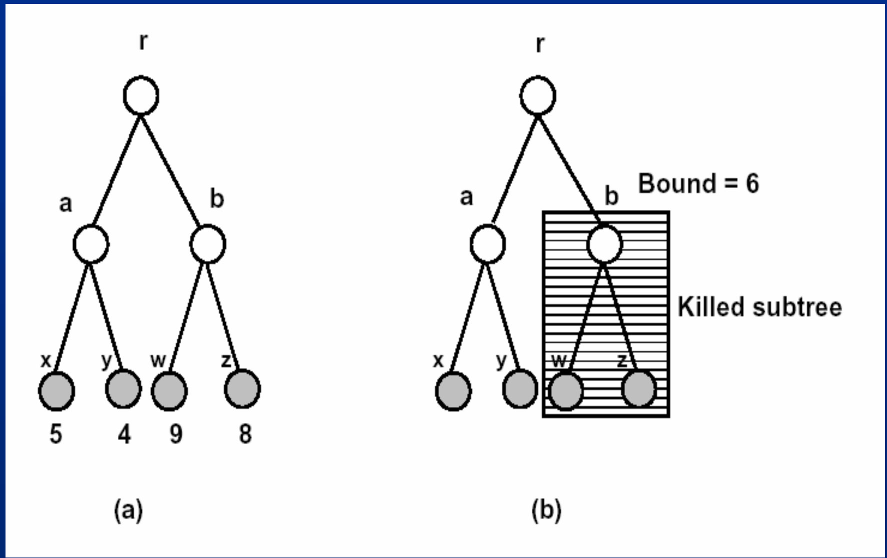
- **Minimum edge cover**
  - Vertex set corresponds to S and edge set to C
  - $A = A_I$ and $c = 1$.
- A minimum cover is {a, b, d} or $x=[11010]^T$
- Let $c=[1, 2, 1, 1, 1]^T$; a minimum cover is {a, c, d}, $x=[10110]^T$

Covering Problem Formulated as Satisfiability Problem

- **Associate a selection variable with each group (element of C)**
- **Associate a clause with each element of S**
  - Each clause represents those groups that can cover the element
  - Disjunction of variables corresponding to groups
- **Note that the product of clauses is a unate expression**
  - Unate cover
- **Edge-cover example**
  - $(x1+x3)(x1+x2+x5)(x2+x3+x5)(x4)(x2+x3+x4)=1$
  - $(x1+x3)$ denotes vertex v1 must be covered by edge a or c
  - $x=[11010]^T$ satisfies the product of sums expression

Branch and Bound Algorithm …

- **Tree search of the solution space**
  - Potentially exponential search.
- **For each branch, a lower bound is computed for all solutions in subtree.**
- **Use bounding function**
  - If the lower bound on the solution cost that can be derived from a set of future choices exceeds the cost of the best solution seen so far
    - **Kill the search**.
- **Good pruning may reduce run-time.**



… Branch and Bound Algorithm

```
BRANCH AND BOUND {
    Current best = anything; Current cost = ∝ ; S = s0;
    while (S ≠ 0) do {
        Select an element  s ∈ S; Remove s from S ;
        Make a branching decision based on s yielding sequences {s_i, i = 1, 2, … , m};
        for ( i = 1 to m) {
            Compute the lower bound b_i of s_i;
            if (b_i ≥ Current cost) Kill s_i;
            else  {
                if (s_i is a complete solution )&(cost of s_i < Current cost) {
                    Current best = s_i; Current cost = cost of s_i ;
                } else if (s_i is not a complete solution ) Add s_i to set S;
            }
        }
    }
}
```

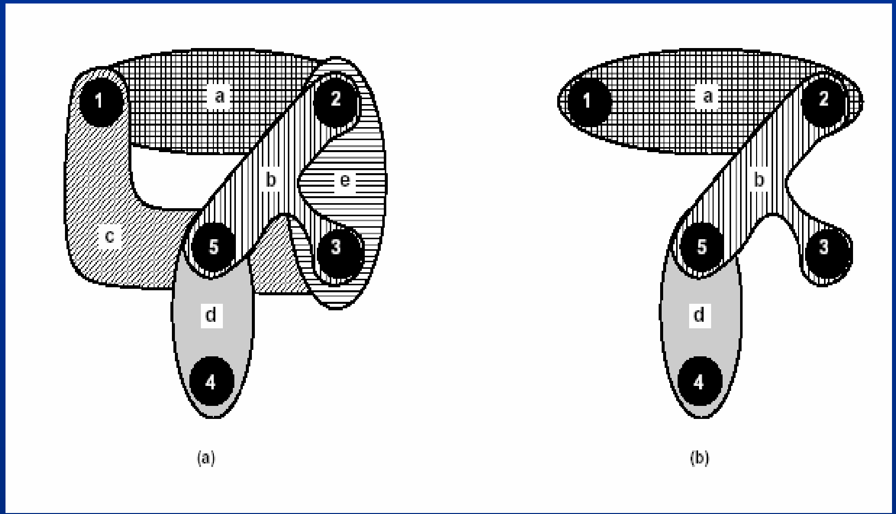
- *S denotes a solution or group of solutions with a subset of decisions made*
- *s0 denotes the sequence of zero length corresp. to initial state with no decisions made*

Covering Reduction Strategies ...

- **Partitioning**
  - If A is block diagonal
    - Solve covering problem for corresponding blocks.
- **Essentials**
  - Column incident to one (or more) rows with single 1
    - Select column,
    - Remove covered row(s) from table.
- Column dominance
  - **If $a_{ki} \geq a_{kj}$ $\forall$ k: remove column j.**
  - **Dominating column covers more rows.**
- Row dominance
  - **If $a_{ik} \geq a_{jk}$ $\forall$ k : remove row i.**
  - **A cover for the dominated rows is a cover for the set.**

… Covering Reduction Strategies

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$



- Fourth column is essential.
- Fifth column is dominated by second column.
- Fifth row dominates fourth row.

Reduced matrix $A = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$

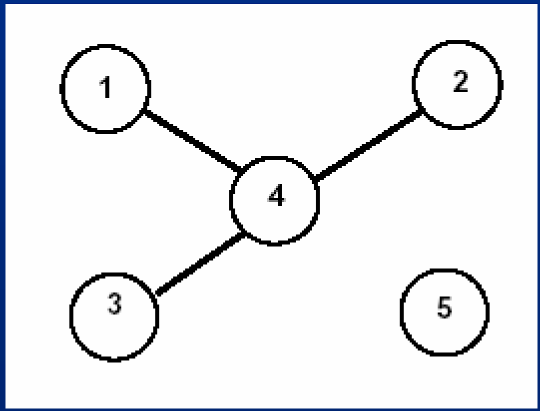
Branch and Bound Exact Covering Algorithm

```
EXACT_COVER(A, x, b) {
    Reduce matrix A and update corresponding x;
    if (Current estimate ≥ |b|) return(b);
    if ( A has no rows ) return (x);
    Select a branching column c;
    x_c = 1 ;
    A~ = A after deleting c and rows incident to it;
    x~ = EXACT_COVER(A~ , x, b);
    if ( |x~| < |b|)    b = x~ ;
    x_c = 0 ;
    A~ = A after deleting c ;
    x~ = EXACT_COVER(A~ , x, b);
    if ( |x~| < |b|)    b = x~ ;
    return (b);
}
```

x contains current solution initially set to 0;
b contains best solution initially set to 1;

Bounding function …

- Estimate lower bound on the covers derived from the current x.
- The sum of 1's in x, plus **bound** on cover for local A
  - **Independent set of rows: no 1 in same column.**
  - **Build graph denoting pairwise independence.**
  - **Find clique number (i.e. largest clique)**
  - **Approximation (lower) is acceptable.**



$$A_l = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

- Row 4 independent from 1,2, 3
- Clique number is 2; Bound is 2

… Bounding function

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

- **There are no independent rows.**
- **Clique number is 1 (1 vertex).**
- **Bound is 1 + 1 (already selected essential).**
- Choose first column x_1
  - **Recur with $A^{\sim}$ = [11].**
  - **Delete one dominated column.**
  - **Take other col. (essential); assume it x_2**
- New cost is 3; $x=[11010]^T$ and $b=[11010]^T$
- Exclude first column x_1
  - **Both columns are essential**
  - **$x=[01110]^T$; cost is 3 (discarded)**
- Returned solution is $x=[11010]^T$

$$A^{\sim} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$