LOGIC SYNTHESIS AND TWO-LEVEL LOGIC OPTIMIZATION

©*Giovanni De Micheli*

Stanford University

Outline

- Overview of logic synthesis.
- Combinational-logic design:
  - Background.
  - Two-level forms.
- Exact minimization.
- Covering algorithms.
- Boolean relations.

- Determine microscopic structure of the circuit.

- Explore *(area-delay)*trade-off:
  - Combinational circuits:
    - I/O delay.
  - Sequential circuits:
    - *cycle-time*.

- Explore *(power-delay)*trade-off:

- Enhance circuit *testability.*

Circuit implementation issues

- Implementation styles:
  - Two-level (e.g. PLA macro cells).
  - Multi-level (e.g. cell-based, array-based).
- Operation:
  - Combinational.
  - Sequential:
    - Synchronous
    - Asynchronous.

Design flow in logic synthesis

- Circuit capture:
  - Tabular specifications of functions or *finite-state machines* (FSMs).
  - Schematic capture.
  - Hardware Description Languages (HDLs).

- Synthesis and optimization:
  - Map circuit representation to abstract model.
  - Transformations on abstract model.
  - Library binding.

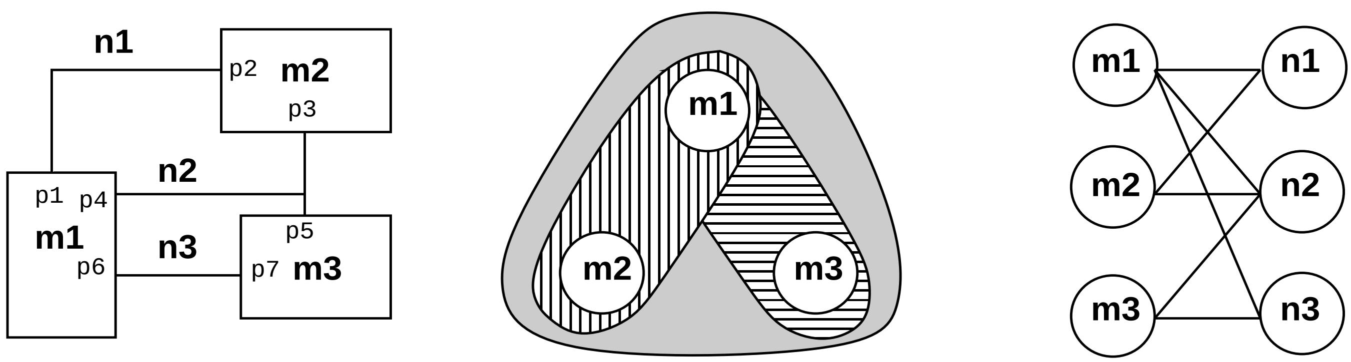
Abstract models

- Models based on graphs.
- Useful for:
  - Machine-level processing.
  - Reasoning about properties.
- Derived from language models by compilation.

Structural views

- Netlists:
  - Modules, nets, incidence.
  - Ports.
  - Hierarchy.
- Incidence (sparse) matrix of a graph.

Example



(a) (b) (c)

Logic funcions

- Black-box model of a combinational module.
- Defined on Boolean Algebra.
- *Support variables* correspond to module inputs.
- Logic functions may have multiple outputs and be *incompletely specified.*

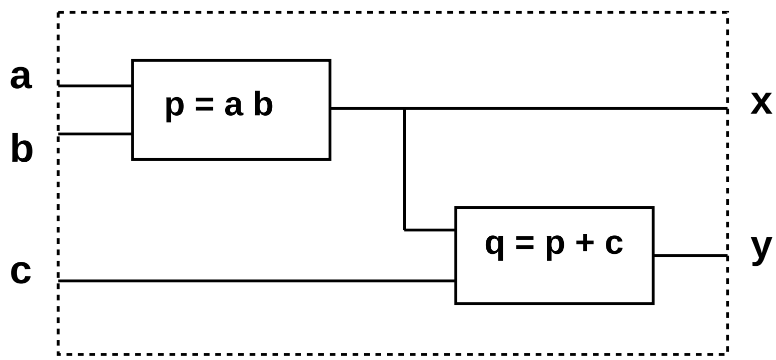
Logic networks

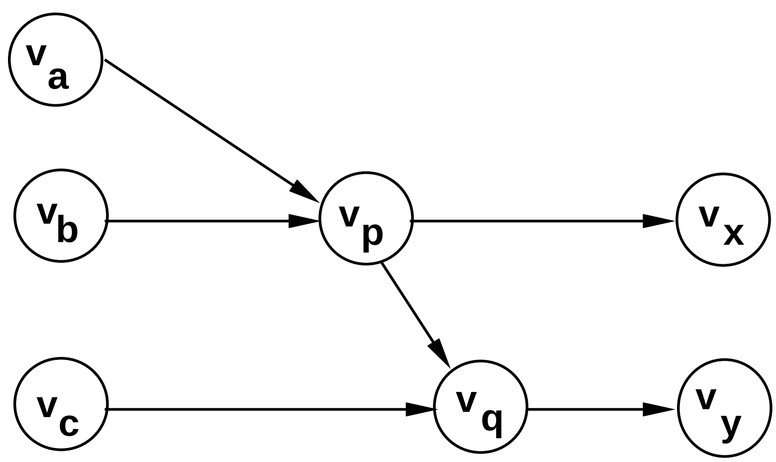
- Mixed structural/behavioral views.

- Useful for multiple-level logic (combinational and sequential).

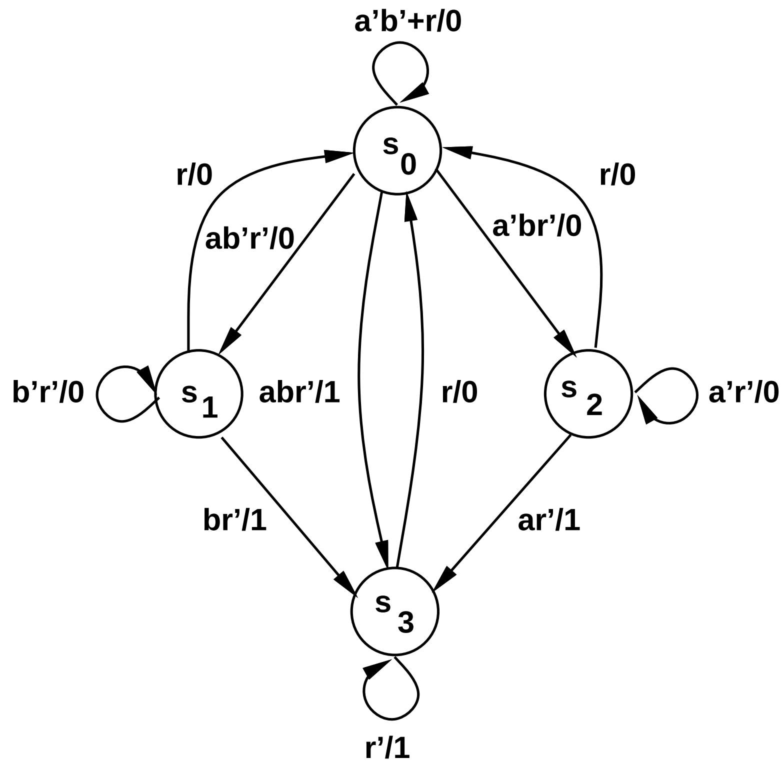
- Interconnection of modules:
  - Logic gates.
  - Logic functions.

Example





State diagrams



- Model behavior of sequential circuits.

- Graph:
  - Vertices = states.
  - Edges = transitions.

Major logic synthesis problems

- Optimization of logic function representation.
  - Minimization of two-level forms.
  - Optimization of Binary Decision Diagrams (BDDs).
- Synthesis of combinational multiple-level logic networks.
  - Optimization or area, delay, power, testability.
- Optmization of FSM models.
  - State minimization, encoding.
- Synthesis of sequential multiple-level logic networks.
  - Optimization or area, delay, power, testability.
- Library binding.
  - Optimal selection of library cells.

Combinational logic design background

- Boolean algebra:
  - Quintuple $(B, +, \cdot, 0, 1)$
  - Binary Boolean algebra $B = \{0, 1\}$

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

The *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.

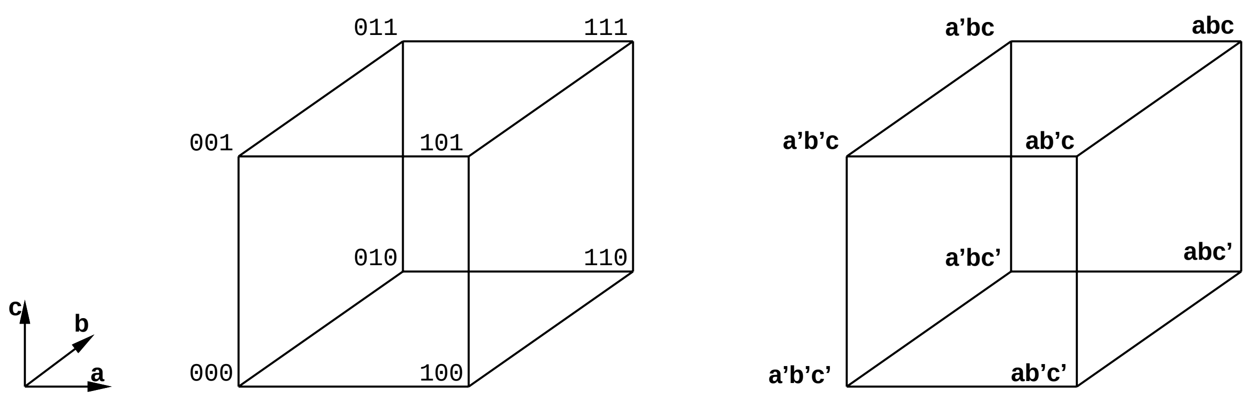
  - $OFF - set$: subset of the domain such that f is false.

  - $DC - set$: subset of the domain such that f is a *don't care* .

- Multiple-output function:

  - Defined for each component.

Cubical representation



Definitions

- Boolean *variables.*

- Boolean *literal*: variable and 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.

Tabular representations

- *Truth table*:
  - List of all minterms of a function.
- *Implicant table* or *cover*:
  - List of implicants of a function sufficient to define function.
- Remark:
  - Implicant tables are smaller in size.

Example of truth table

$$x = ab + a'c; \quad y = ab + bc + ac$$

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

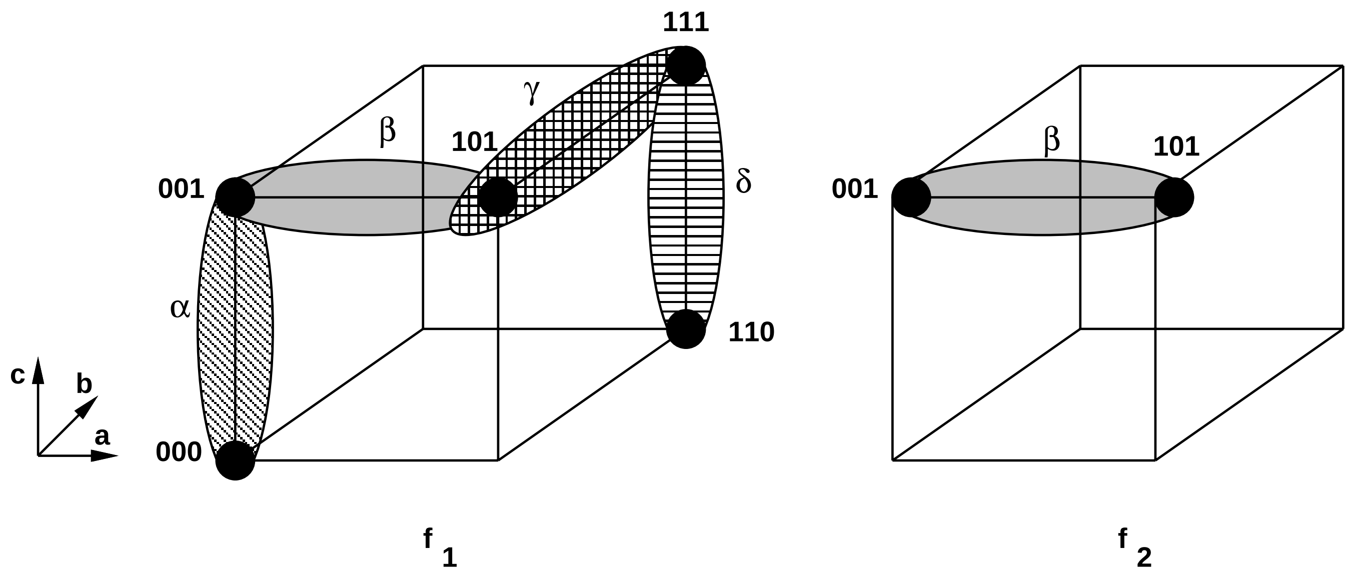
Example of implicant table

$$x = ab + a'c; \quad y = ab + bc + ac$$

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

Cubical representation of minterms and implicants

© GDM



- $f_1 = a'b'c' + a'b'c + ab'c + abc + abc'$

- $f_2 = a'b'c + ab'c$

Two-level logic optimization motivation

- Reduce size of the representation.

- Direct implementation:

  – PLAs – reduce size and delay.

- Other implementation styles
  (e.g. multi-level):

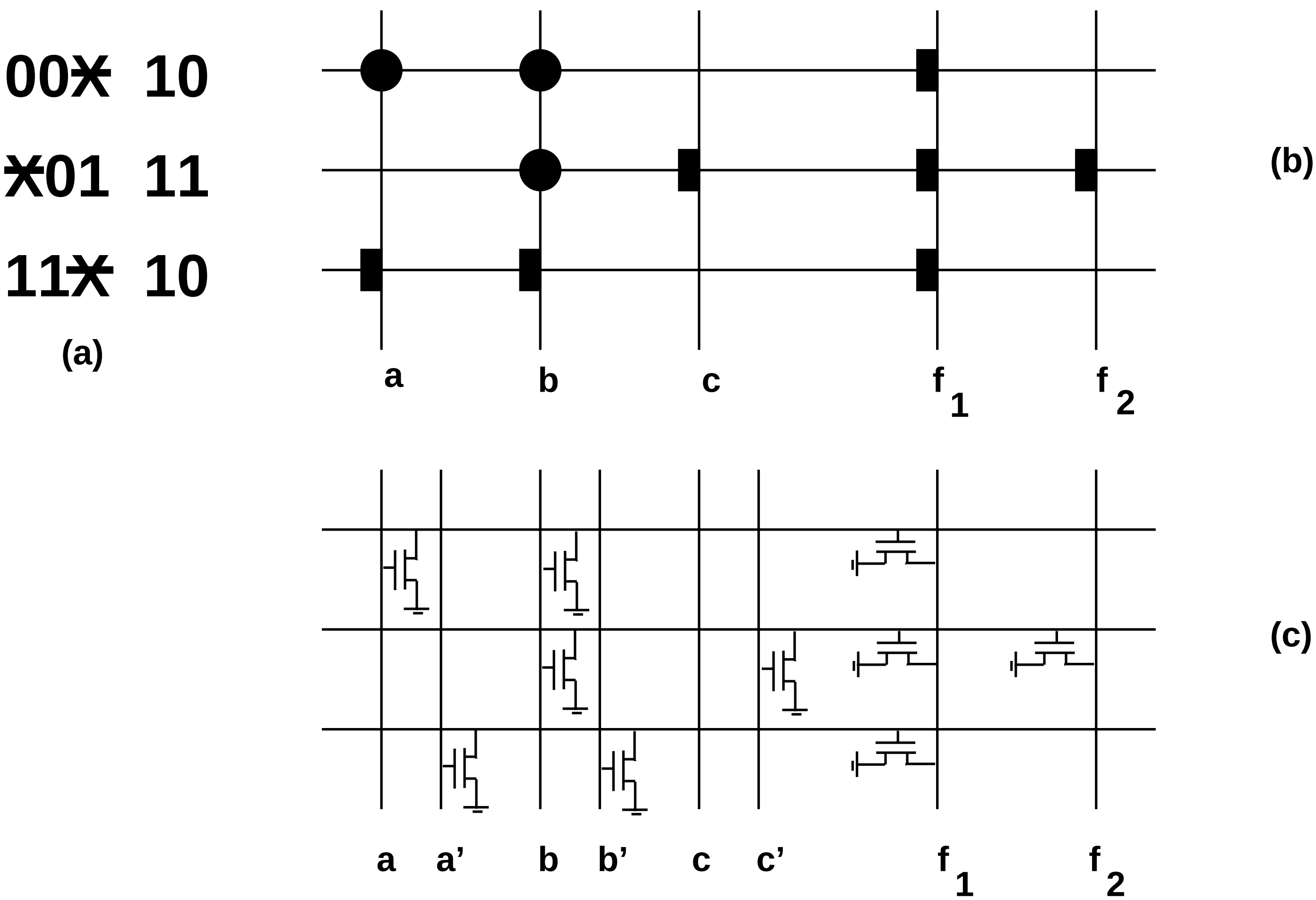
  – Reduce amount of information.

  – Simplify local functions and connections.

Programmable logic arrays

- Macro-cells with rectangular structure.
- Implement any multi-output function.
- Layout easily generated by module generators.
- Fairly popular in the seventies/eighties (NMOS).
- Still used for control-unit implementation.

Programmable logic array



- $f_1 = a'b' + b'c + ab \qquad f_2 = b'c$

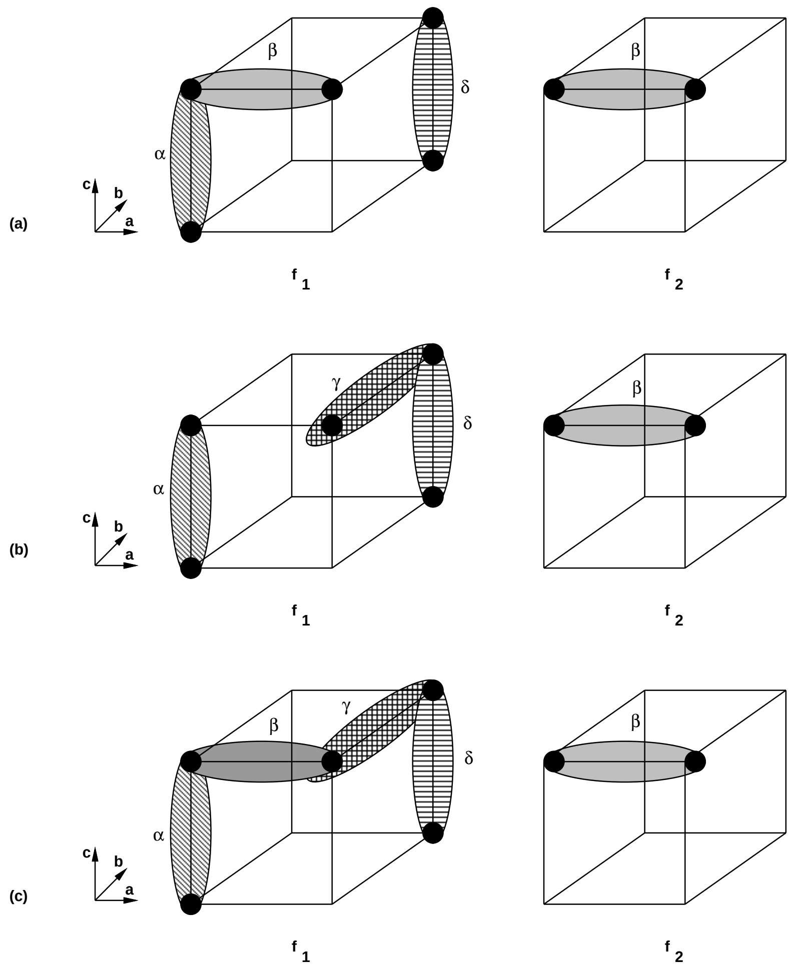
Two-level optimization

- Assumptions:
  - Primary goal is to reduce the number of implicants.
  - All implicants have the same cost.
  - Secondary goal is to reduce the number of literals.
- Rationale:
  - Implicants correspond to PLA rows.
  - Literals correspond to transistors.

Definitions

- *Minimum cover*:
  - Cover of the function with minimum number of implicants.
  - Global optimum.
- *Minimal cover* or *irredundant cover*:
  - Cover of the function that is not a proper superset of another cover.
  - No implicant can be dropped.
  - Local optimum.
- *Minimal cover w.r.t. 1-implicant containment.*
  - No implicant is contained by another one.
  - Weak local optimum.

Example



- $f_1 = a'b'c' + a'b'c + ab'c + abc + abc'$

- $f_2 = a'b'c + ab'c$

Definitions

- *Prime* implicant:
  - Implicant not contained by any other implicant.
- *Prime* cover:
  - Cover of prime implicants.
- *Essential* prime implicant:
  - There exist some minterm covered only by that prime implicant.

Logic minimization

- *Exact* methods:
  - Compute minimum cover.
  - Often impossible for large functions.
  - Based on *Quine McCluskey* method.

- *Heuristic* methods:
  - Compute minimal covers
    (possibly minimum).
  - Large variety of methods and programs:
    - MINI, PRESTO, ESPRESSO.

Exact logic minimization

- *Quine's theorem:*
  - There is a minimum cover that is prime.
- Consequence:
  - Search for minimum cover can be restricted to prime implicants.
- Quine McCluskey method:
  - Compute prime implicants.
  - Determine minimum cover.

Prime implicant table

- Rows: minterms.

- Columns: prime implicants.

- Exponential size:
  - 2^n minterms.
  - Up to $3^n/n$ prime implicants.

- Remark:
  - Some functions have much fewer primes.
  - Minterms can be grouped together.

Example

- Function: $f = a'b'c' + a'b'c + ab'c + abc + abc'$

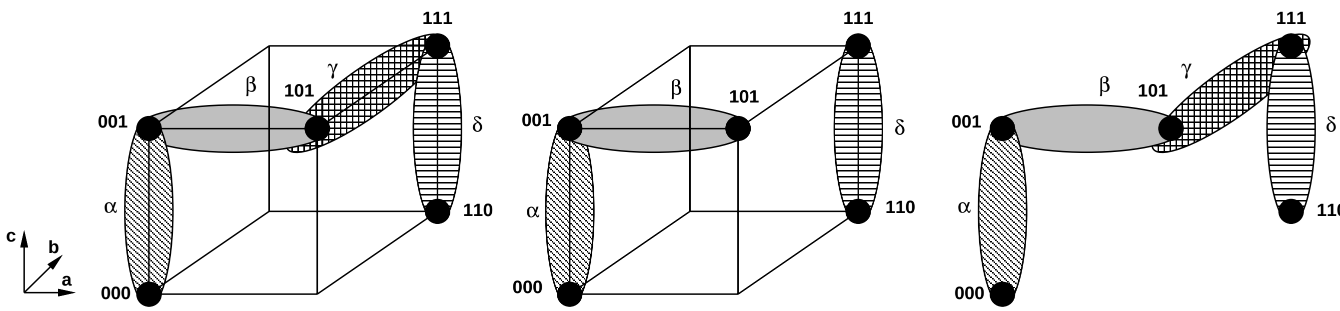
- Primes:

α	00*	1
β	*01	1
γ	1*1	1
δ	11*	1

- Implicant table:

	α	β	γ	δ
000	1	0	0	0
001	1	1	0	0
101	0	1	1	0
111	0	0	1	1
110	0	0	0	1

Example


111
β
101
γ
001
δ
α
110
c
b
a
000
111
β
101
001
δ
α
110
000
111
β
γ
101
001
δ
α
110
000

(a) (b) (c)

Minimum cover
early methods

- Reduce table:
  - Iteratively identify essentials,
    save them in the cover,
    remove covered minterms.

- *Petrick*'s method.
  - Write covering clauses in *pos* form.
  - Multiply out *pos* form into *sop* form.
  - Select cube of minimum size.
  - *Remark*:
    * Multiplying out clauses is exponential.

Example
Petrick's method

- *pos* clauses:
  - $(\alpha)(\alpha+\beta)(\beta+\gamma)(\gamma+\delta)(\delta)=1$
- *sop* form:
  - $\alpha\beta\delta+\alpha\gamma\delta=1$
- Solutions:
  - $\{\alpha,\beta,\delta\}$
  - $\{\alpha,\gamma,\delta\}$

Matrix representation

- View table as Boolean matrix: $\mathbf{A}$.

- Selection Boolean vector for primes: $\mathbf{x}$.

- Determine $\mathbf{x}$ such that:

  - $\mathbf{A}\ \mathbf{x}\ \geq \mathbf{1}$.

  - Select enough columns to cover all rows.

- Minimize cardinality of $\mathbf{x}$:

  - Example: $\mathbf{x} = [1101]^T$

Example

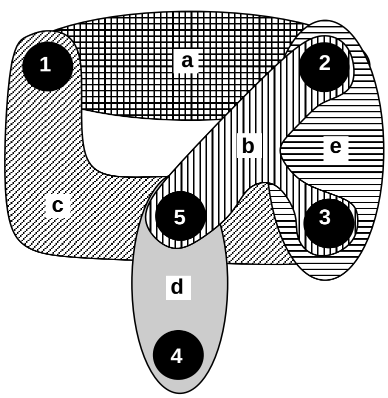
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Covering problem

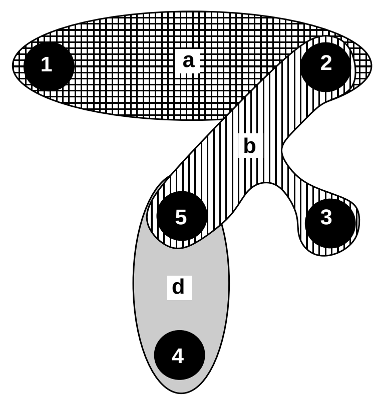
- Set covering problem:
  - A set S. (Minterm set).
  - A collection C of subsets. (Implicant set).
  - Select fewest elements of C to cover S.
- Intractable.
- Exact method:
  - Branch and bound algorithm.
- Heuristic methods.

Example
edge-cover of a hypergraph

© GDM



(a)



(b)

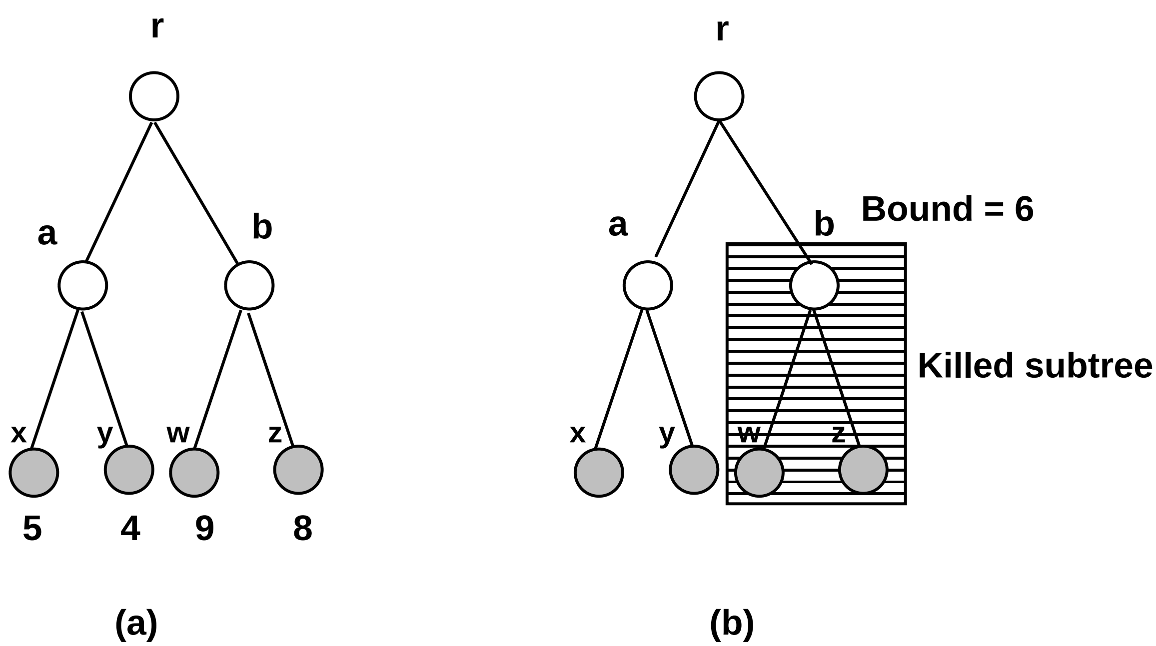
Branch and bound algorithm

- Tree search of the solution space:
  - Potentially exponential search.
- 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 = s_0;
    while (S ≠ ∅) do {
        Select an element in 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 ) {
                    Current_best = s_i;
                    Current_cost = cost of s_i ;
                }
                else
                    Add s_i to set S;
            }
        }
    }
}
```

Example



- Partitioning:
  - If **A** is block diagonal:
    - Solve covering problem for corresponding blocks.

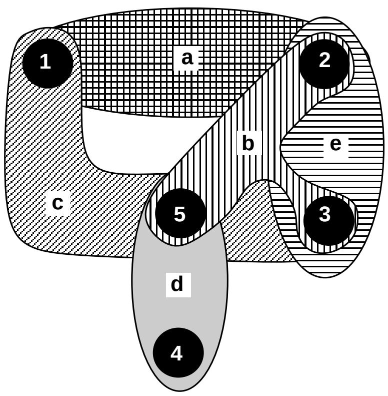
- Essentials (EPI):
  - Column incident to
    one (or more) row with single 1:
    - Select column.
    - Remove covered row(s) from table.

Branch and bound algorithm for covering
Reduction strategies

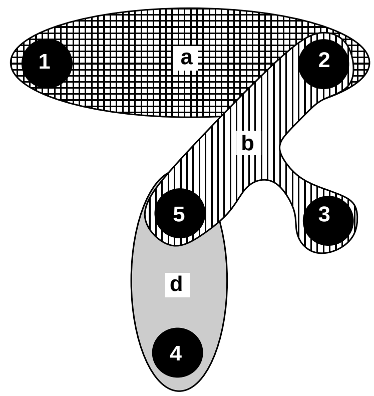
- Column (implicant) dominance:
  - If $a_{ki} \geq a_{kj} \; \forall k$:
    - remove column j.

- Row (minterm) dominance:
  - If $a_{ik} \geq a_{jk} \; \forall k$:
    - Remove row i.

Example



(a)



(b)

$$
\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}
$$

Example reduction

- Fourth column is essential.
- Fifth column is dominated.
- Fifth row is dominant.
- $\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

Branch and bound covering algorithm

$EXACT\_COVER(\mathbf{A}, \mathbf{x}, \mathbf{b})$ {

Reduce matrix $\mathbf{A}$ and update corresponding $\mathbf{x}$;

if $(Current\_estimate \geq |\mathbf{b}|)$ **return**$(\mathbf{b})$;

if ($\mathbf{A}$ has no rows) **return** $(\mathbf{x})$;

Select a branching column c;

$x_c = 1$;

$\widetilde{\mathbf{A}} = \mathbf{A}$ after deleting c and rows incident to it;

$\tilde{\mathbf{x}} = EXACT\_COVER(\widetilde{\mathbf{A}}, \mathbf{x}, \mathbf{b})$;

if ($|\tilde{\mathbf{x}}| < |\mathbf{b}|$)

$\mathbf{b} = \tilde{\mathbf{x}}$;

$x_c = 0$;

$\widetilde{\mathbf{A}} = \mathbf{A}$ after deleting c ;

$\tilde{\mathbf{x}} = EXACT\_COVER(\widetilde{\mathbf{A}}, \mathbf{x}, \mathbf{b})$;

if ($|\tilde{\mathbf{x}}| < |\mathbf{b}|$)

$\mathbf{b} = \tilde{\mathbf{x}}$;

return $(\mathbf{b})$;

}

Bounding function

- Estimate lower bound on the covers derived from the current **x**.

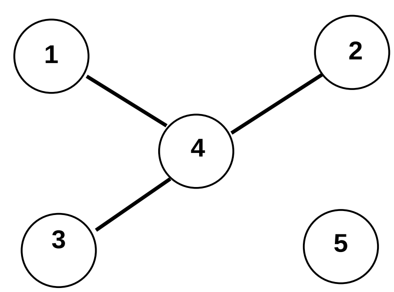
- The sum of the ones in **x**, plus bound on cover for local **A**:
  - Independent set of rows:
    - No 1 in same column.
  - Build graph denoting pairwise independence.
  - Find clique number.
  - Approximation (lower) is acceptable.

Example

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

- Row 4 independent from 1,2,3.

- Clique number is 2.



- Bound is 2.

Example

- $\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

- There are no independent rows.

- Clique number is 1 (one vertex).

- Bound is $1 + 1$ (already selected essential).

Example

- $\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$

- Choose first column:
  - Recur with $\widetilde{\mathbf{A}} = [11]$.
    * Delete one dominated column.
    * Take other column (essential).
  - New cost is 3.

- Exclude first column:
  - Find another solution with cost 3 (discarded).

ESPRESSO-EXACT

- Exact minimizer [Rudell].

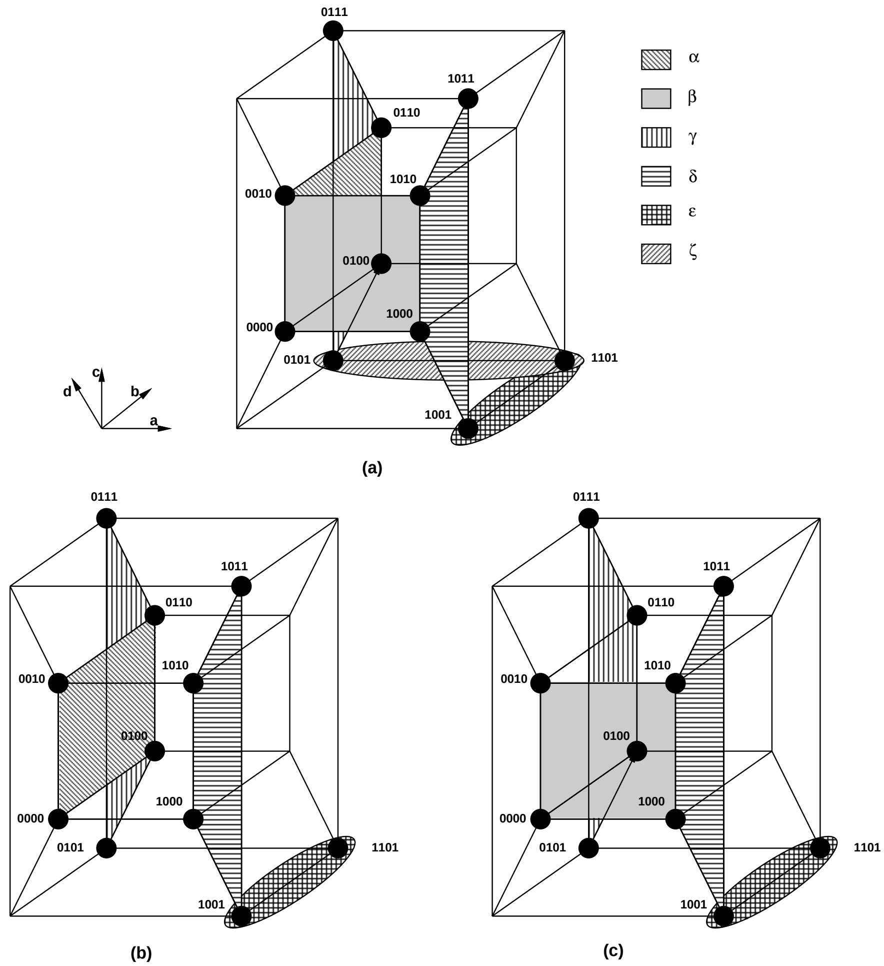
- Exact *branch and bound* covering.

- Compact implicant table:

  - Group together minterms
    covered by the same implicants.

- Very efficient. Solves most problems.

Example



α	0**0	1
β	*0*0	1
γ	01**	1
δ	10**	1
ϵ	1*01	1
ζ	*101	1

Example
Prime implicant table
(after removing essentials)

	α	β	ϵ	ζ
0000,0010	1	1	0	0
1101	0	0	1	1

Recent developments

- Many minimization problems can be solved exactly today.

- Usually bottleneck is table size.

- Implicit representation of prime implicants:
  - Methods based on BDDs [COUDERT]:
    - To represent sets.
    - To do dominance simplification.
  - Methods based on signature cubes [MCGEER]
    - Represent set of primes.

Summary
Exact two-level minimization of logic functions

- Based on derivatives of Quine-McCluskey method.

- Many minimization problems can be now solved exactly.

- Usual problems are memory size and time.

Boolean relations

- Generalization of Boolean functions.

- More than one output pattern may correspond to an input pattern.

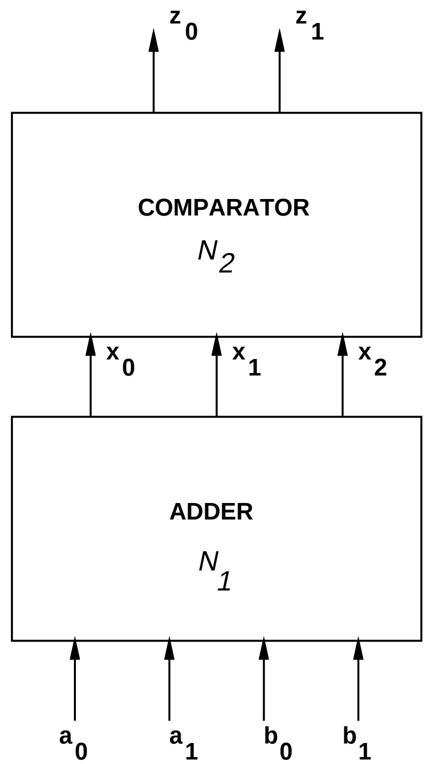
- Some *degrees of freedom* in finding an implementation:

  - More general than *don't care* conditions.

- Problem:

  - Given a Boolean relation,
    find minimum cover of a compatible function.

Example



- Compare:
  - $\mathbf{a} + \mathbf{b} > 4$?
  - $\mathbf{a} + \mathbf{b} < 3$?

Example

a_1	a_0	b_1	b_0	**x**
0	0	0	0	{ 000, 001, 010 }
0	0	0	1	{ 000, 001, 010 }
0	0	1	0	{ 000, 001, 010 }
0	1	0	0	{ 000, 001, 010 }
1	0	0	0	{ 000, 001, 010 }
0	1	0	1	{ 000, 001, 010 }
0	0	1	1	{ 011, 100 }
0	1	1	0	{ 011, 100 }
1	0	0	1	{ 011, 100 }
1	0	1	0	{ 011, 100 }
1	1	0	0	{ 011, 100 }
0	1	1	1	{ 011, 100 }
1	1	0	1	{ 011, 100 }
1	0	1	1	{ 101, 110, 111 }
1	1	1	0	{ 101, 110, 111 }
1	1	1	1	{ 101, 110, 111 }

Example (2)
Minimum implementation

a_1	a_0	b_1	b_0	**x**
0	*	1	*	010
1	*	0	*	010
1	*	1	*	100
*	*	*	1	001
*	1	*	*	001

- Remark:
  - Circuit is no longer an adder.

Minimization of Boolean relations

- Since there are many possible output values there are many logic functions implementing the relation.
  - Compatible functions.
- Find a function with minimum cardinality.
- Do not enumerate all possible functions:
  - May be too many.
- Represent the primes of all possible functions:
  - Compatible primes ($c - primes$).

Minimization of Boolean relation

- Exact:
  - Find a set of compatible primes.
  - Solve a *binate* covering problem.
    * Consistency relations.

- Heuristic:
  - Iterative improvement [GYOCRO].

Example

- Boolean relation:

0	0	0	{ 00 }
0	0	1	{ 00 }
0	1	0	{ 00 }
0	1	1	{ 10 }
1	0	0	{ 00 }
1	0	1	{ 01 }
1	1	0	{ 00,11 }
1	1	1	{ 00,11 }

- Compatible primes:

α	0	1	1	10
β	1	0	1	01
γ	1	1	0	11
δ	1	1	1	11
ϵ	*	1	1	10
ζ	1	*	1	01
η	1	1	*	11

Example

- Input 011 – output 10.
  - Covering clause $(\alpha + \epsilon)$.
- Input 111 – output 00 or 11.
  - No implicant – 00 – correct.
  - Either η or $\epsilon \cup \zeta$ – output 11 – correct.
  - Only ϵ or ζ is selected – output 10 or 01 – WRONG.
  - Covering clause $\eta + \epsilon\zeta + \epsilon'\zeta'$ – binate.
- Overall covering clause:

$$(\alpha + \epsilon) \cdot (\beta + \zeta) \cdot (\epsilon + \zeta' + \eta) \cdot (\epsilon' + \zeta + \eta)$$

Binate covering

- Covering problem with *binate clause.*
- Implications:
  - The selection of a prime
    may exclude other primes.
- No guarantee of finding a feasible solution:
  - Inconsistent clauses.
- *Minimum-cost satisfiability* problem.
  - Much harder to solve than unate cover.
  - Branch and bound algorithm.
  - BDD-based methods.

Summary
Boolean relations

- Generalization of Boolean functions.
  - Many possible output patterns.
- Useful for modeling:
  - Cascaded blocks.
  - Portions of multiple-level networks.
- More degree of freedom in implementation.
- Harder problem to solve.