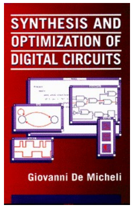
Heuristic Two-level Logic Optimization

Giovanni De Micheli

Integrated Systems Centre

EPF Lausanne



Module 1

- **Objective**
  - **Data structures for logic optimization**
  - **Data representation and encoding**

Some more background

- Function $f\ (x_1, x_2, \ldots., x_i, \ldots., x_n)$
- Cofactor of f with respect to variable x_i
  - $f_{xi} = f\ (x_1, x_2, \ldots., 1, \ldots., x_n)$
- Cofactor of f with respect to variable x_i'
  - $f_{xi'} = f\ (x_1, x_2, \ldots., 0, \ldots., x_n)$
- Boole's expansion theorem:
  - $f\ (x_1, x_2, \ldots., x_i, \ldots., x_n) = x_i\, fx_i + x_{i'}\, fx_i'$
  - Also credited to Claude Shannon

Example

- **Function:** $f = ab + bc + ac$
- **Cofactors:**
  - $f_a = b + c$
  - $f_{a'} = bc$
- **Expansion:**
  - $f = a\, f_a + a' f_{a'} = a(b + c) + a'bc$

Unateness

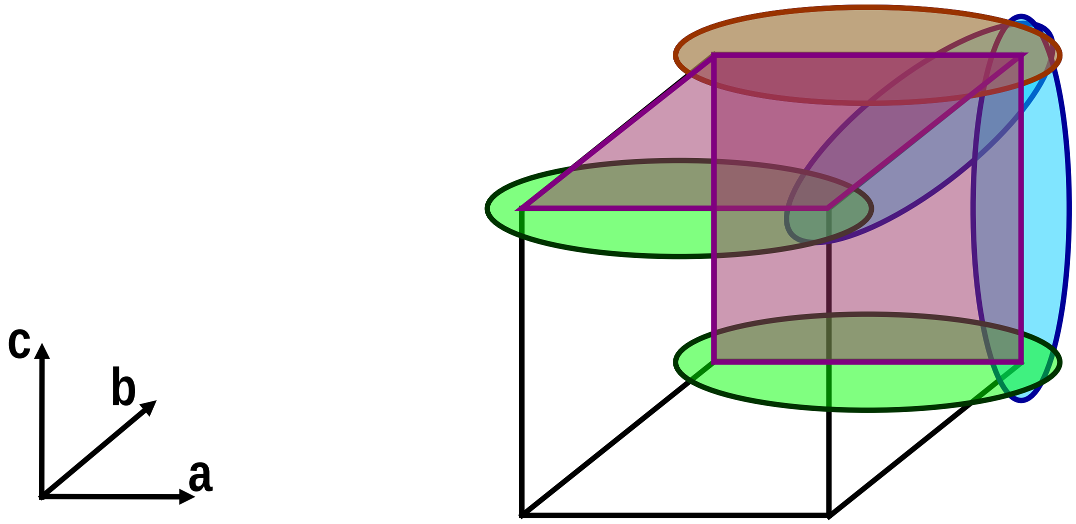
- Function $f\ (x_1, x_2, \ldots., x_i, \ldots., x_n)$
- ***Positive unate*** **in** x_i **when:**
  - $f_{xi} \geq f_{xi'}$
- ***Negative unate*** **in** x_i **when:**
  - $f_{xi} \leq f_{xi'}$
- **A function is positive/negative unate when positive/ negative unate in all its variables**

Operators

- Function $f\,(x_1, x_2, \ldots, x_i, \ldots, x_n)$
- *Boolean difference* of f w.r.t. variable x_i:
  - $\partial f/\partial x_i \equiv f_{x_i} \oplus f_{x_i'}$
- *Consensus* of f w.r.t. variable x_i:
  - $C_{x_i} \equiv f_{x_i} \cdot f_{x_i'}$
- *Smoothing* of f w.r.t. variable x_i:
  - $S_{x_i} \equiv f_{x_i} + f_{x_i'}$

Example
$f = ab + bc + ac$

- The Boolean difference $\partial f/\partial a = f_a \oplus f_{a'} = b'c + bc'$
- The consensus $C_a = f_a \cdot f_{a'} = bc$
- The smoothing $S_a \equiv f_a + f_{a'} = b + c$



Generalized expansion

- **Given:**
  - **A Boolean function f.**
  - **Orthonormal set of functions:**

    ϕ_i, $i = 1, 2, \ldots, k$
- **Then:**
  - $f = \sum_i^k \phi_i \cdot f_{\phi_i}$
  - **Where f_{ϕ_i} is a *generalized cofactor*.**
- **The generalized cofactor is not unique, but satisfies:**
  - $f \cdot \phi_i \subseteq f_{\phi_i} \subseteq f + \phi_i'$

Example

- **Function:** $f = ab + bc + ac$
- **Basis:** $\phi_1 = ab$ **and** $\phi_2 = a' + b'$.
- **Bounds:**
  - $ab \subseteq f_{\phi_1} \subseteq 1$
  - $a'bc + ab'c \subseteq f_{\phi_2} \subseteq ab + bc + ac$
- **Cofactors:** $f_{\phi_1} = 1$ **and** $f_{\phi_2} = a'bc + ab'c$.

$$f = \phi_1 f_{\phi_1} + \phi_2 f_{\phi_2}$$

$$= ab1 + (a' + b')(a'bc + ab'c)$$

$$= ab + bc + ac$$

Generalized expansion theorem

- **Given:**
  - Two function f and g.
  - Orthonormal set of functions: ϕ_i , $i=1,2,\ldots,k$
  - Boolean operator $\odot$
- **Then:**
  - $f \odot g = \sum_i^k \phi_i \cdot (f_{\phi_i} \odot g_{\phi_i})$
- **Corollary:**
  - $f \odot g = x_i \cdot (f_{x_i} \odot g_{x_i}) + x_i' \cdot (f_{x_i'} \odot g_{x_i'})$

Matrix representation of logic covers

- **Representations used by logic minimizers**
- **Different formats**
  - **Usually one row per implicant**
- **Symbols:**
  - **0, 1, * , ...**
- **Encoding:**

∅	00
0	10
1	01
*	11

Advantages of positional cube notation

- Use binary values:
  - Two bits per symbols
  - More efficient than a byte (char)
- Binary operations are applicable
  - Intersection – bitwise AND
  - Supercube – bitwise OR
- Binary operations are very fast and can be parallelized

Example

- **f = a'd' + a'b + ab' + ac'd**

10	11	11	10
10	01	11	11
01	10	11	11
01	11	10	01

Cofactor computation

- Cofactor of α w.r. to β
  - Void when α does not intersect β
  - $a_1 + b_1' \quad a_2 + b_2' \quad \ldots \quad a_n + b_n'$
- Cofactor of a set $C = \{\gamma_i\}$ w.r. to β:
  - Set of cofactors of γ_i w.r. to β

Example f = a'b' + ab

- **Cofactor w.r.t. to 01 11**
  - First row – void
  - Second row – 11 01
- **Cofactor $f_a = b$**

10 10
01 01

00 00
01 11

---

01 11 → void
10 00

---

11 01

Multiple-valued-input functions

- Input variables can take many values
- Representations:
  - Literals: set of valid values
  - Function = sum of products of literals
- Positional cube notation can be easily extended to mvi
- Key fact
  - Multiple-output binary-valued functions represented as mvi single-output functions

Example

- **2-input, 3-output function:**
  - $f_1 = a'b' + ab$
  - $f_2 = ab$
  - $f_3 = ab' + a'b$
- **Mvi representation:**

10	10	100
10	01	001
01	10	001
01	01	110

Module 2

- **Objective**
  - **Operations on logic covers**
  - **Application of the recursive paradigm**
  - **Fundamental mechanisms used inside minimizers**

Operations on logic covers

- Recursive paradigm
  - Expand about a mv-variable
  - Apply operation to co-factors
  - Merge results
- Unate heuristics
  - Operations on unate functions are simpler
  - Select variables so that cofactors become unate functions
- Recursive paradigm is general and applicable to different data structures
  - Matrices and binary decision diagrams

Tautology

- **Check if a function is always TRUE**
- **Recursive paradigm:**
  - **Expend about a mvi variable**
  - **If all cofactors are TRUE, then the function is a tautology**
- **Unate heuristics**
  - **If cofactors are unate functions, additional criteria to determine tautology**
  - **Faster decision**

Recursive tautology

- **TAUTOLOGY:**
  - **The cover matrix has a row of all 1s. (Tautology cube)**
- **NO TAUTOLOGY:**
  - **The cover has a column of 0s. (A variable never takes a value)**
- **TAUTOLOGY:**
  - **The cover depends on one variable, and there is no column of 0s in that field**
- **Decomposition rule:**
  - **When a cover is the union of two subcovers that depend on disjoint sets of variables, then check tautology in both subcovers**

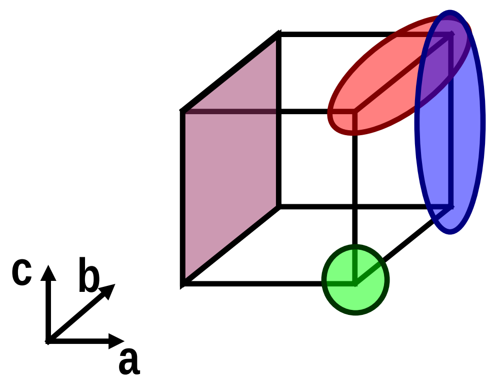
Example
f = ab + ac + ab’c’ + a’

- Select variable a
- Cofactor w.r.t. to a’ is

  11 11 11 – Tautology.

- Cofactor w.r.t. to a is:

11	01	11
11	11	01
11	10	10



01	01	11
01	11	01
01	10	10
10	11	11
00	11	11
01	01	11
01	11	01
01	10	10
00	11	11
00	00	00
11	01	11
11	11	01
11	10	10

Example (2)

11	01	11
11	11	01
11	10	10

- **Select variable b.**
- **Cofactor w.r. to b' is**

11	11	01
11	11	10

- **No column of 0 - Tautology**
- **Cofactor w.r. to b is:**

11	11	11

- **Function is a *TAUTOLOGY*.**

11	01	11
11	11	01
11	10	10
11	00	11
11	00	11
11	10	01
11	10	10
00	00	00
11	11	01
11	11	10

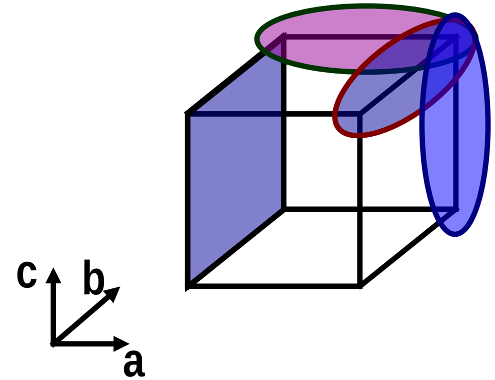
Containment

- **Theorem:**
  - **A cover F contains an implicant α if and only if F_{α} is a tautology**
- **Consequence:**
  - **Containment can be verified by the tautology algorithm**

Example
f = ab + ac + a'

- Check covering of bc : 11 01 01.
- Take the cofactor:

01	11	11
01	11	11
10	11	11



- Tautology – bc is contained by f.

Complementation

- Recursive paradigm
  - $f' = x f'_x + x' f'_{x'}$
- Steps:
  - Select variable
  - Compute co-factors
  - Complement co-factors
- Recur until cofactors can be complemented in a straightforward way

Termination rules

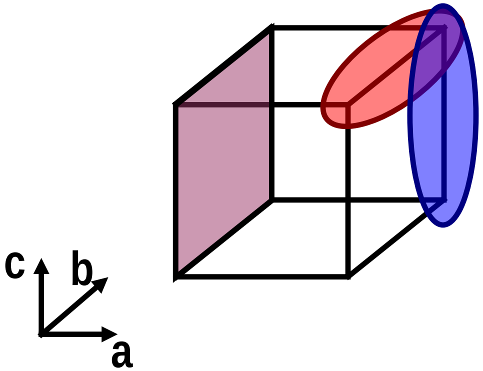
- The cover F is void
  - Hence its complement is the universal cube
- The cover F has a row of 1s
  - Hence F' is a tautology and its complement is void
- The cover F consists of one implicant.
  - Hence the complement is computed by DeMorgan's law
- All implicants of F depend on a single variable, and there is not a column of 0s.
  - The function is a tautology, and its complement is void

Unate functions

- **Theorem:**
  - If f is positive unate in x, then
    - $f' = f'_x + x' f'_{x'}$
  - If f is negative unate in x, then
    - $f' = x f'_x + f'_{x'}$
- **Consequence:**
  - Complement computation is simpler
  - Follow only one branch in the recursion
- **Heuristics**
  - Select variables to make the cofactor unate

Example
f = ab + ac + a'

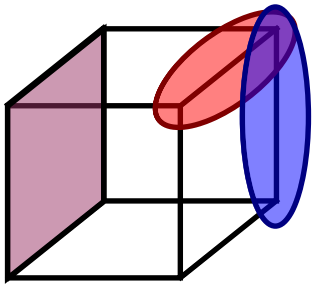
- Select binate variable a



- Compute cofactors:
  - $F_{a'}$ is a tautology, hence $F'_{a'}$ is void.
  - F_a yields:

11	01	11
11	11	01

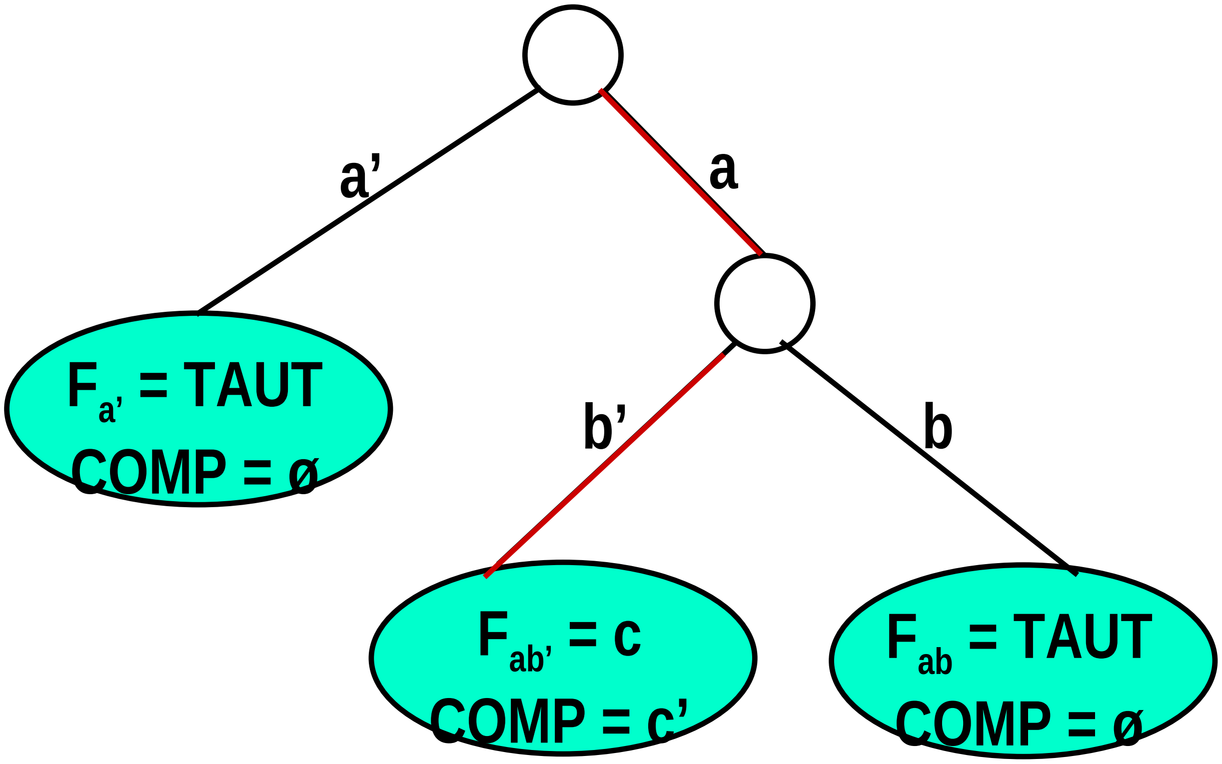
Example (2)



- Select unate variable b
- Compute cofactors:
  - F_{ab} is a tautology, hence F'_{ab} is void
  - $F_{ab'}$ = 11 11 01 and its complement is 11 11 10
- Re-construct complement:
  - 11 11 10 intersected with *Cube*(b') = 11 10 11 yields 11 10 10
  - 11 10 10 intersected with *Cube*(a) = 01 11 11 yields 01 10 10
- Complement: F' = 01 10 10

Example (3)

- **Recursive search:**



Complement: a b'c'

Boolean cover manipulation summary

- Recursive methods are efficient operators for logic covers
  - Applicable to matrix-oriented representations
  - Applicable to recursive data structures like BDDs
- Good implementations of matrix-oriented recursive algorithms are still very competitive
  - Heuristics tuned to the matrix representations

Module 3

- **Objectives**
  - **Heuristic two-level minimization**
  - **The algorithms of ESPRESSO**

Heuristic logic minimization

- Provide irredundant covers with “reasonably small” sizes
- Fast and applicable to many functions
  - Much faster than exact minimization
- Avoid bottlenecks of exact minimization
  - Prime generation and storage
  - Covering
- Motivation
  - Use as internal engine within multi-level synthesis tools

Heuristic minimization -- principles

- **Start from initial cover**
  - **Provided by designer or extracted from hardware language model**
- **Modify cover under consideration**
  - **Make it prime and irredundant**
  - **Perturb cover and re-iterate until a small irredundant cover is obtained**
- **Typically the size of the cover decreases**
  - **Operations on limited-size covers are fast**

Heuristic minimization - operators

- **Expand**
  - **Make implicants prime**
  - **Removed covered implicants**
- **Reduce**
  - **Reduce size of each implicant while preserving cover**
- **Reshape**
  - **Modify implicant pairs: enlarge one and reduce the other**
- **Irredundant**
  - **Make cover irredundant**

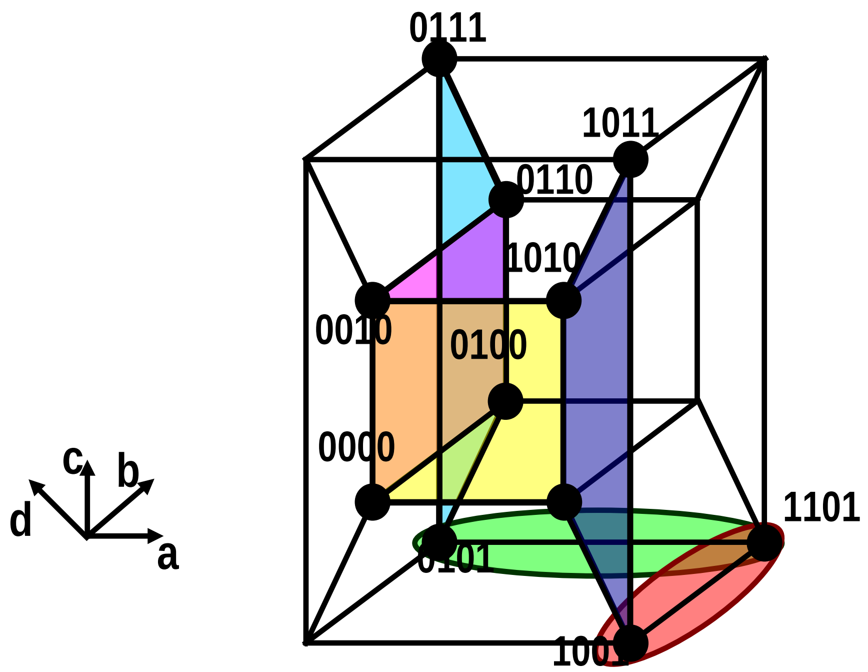
Example

- **Initial cover**
  - **(without positional cube notation)**

0000	1
0010	1
0100	1
0110	1
1000	1
1010	1
0101	1
0111	1
1001	1
1011	1
1101	1

Example

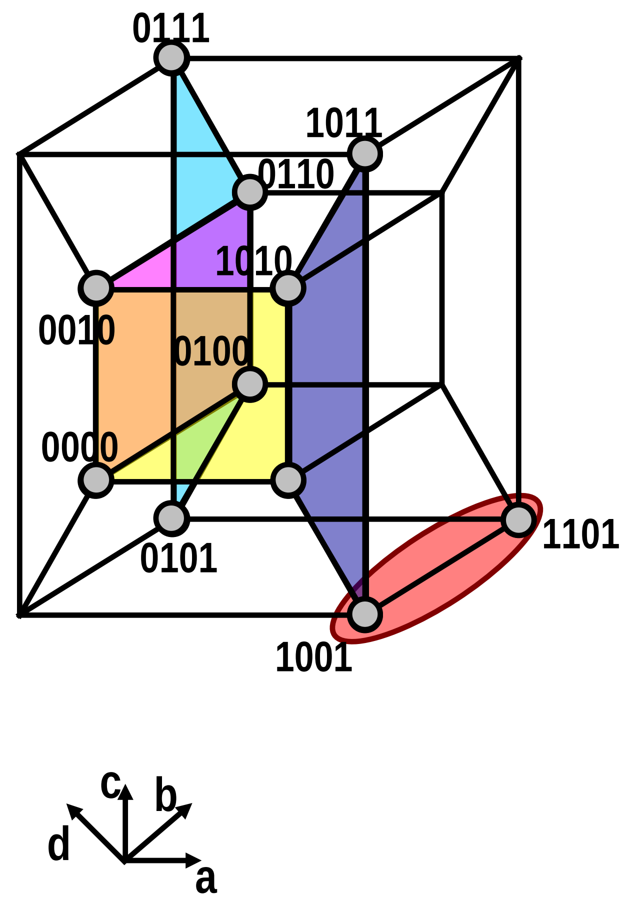
- **Set of primes**



α	0 * * 0	1
β	* 0 * 0	1
γ	0 1 * *	1
δ	1 0 * *	1
ε	1 * 0 1	1
ζ	* 1 0 1	1

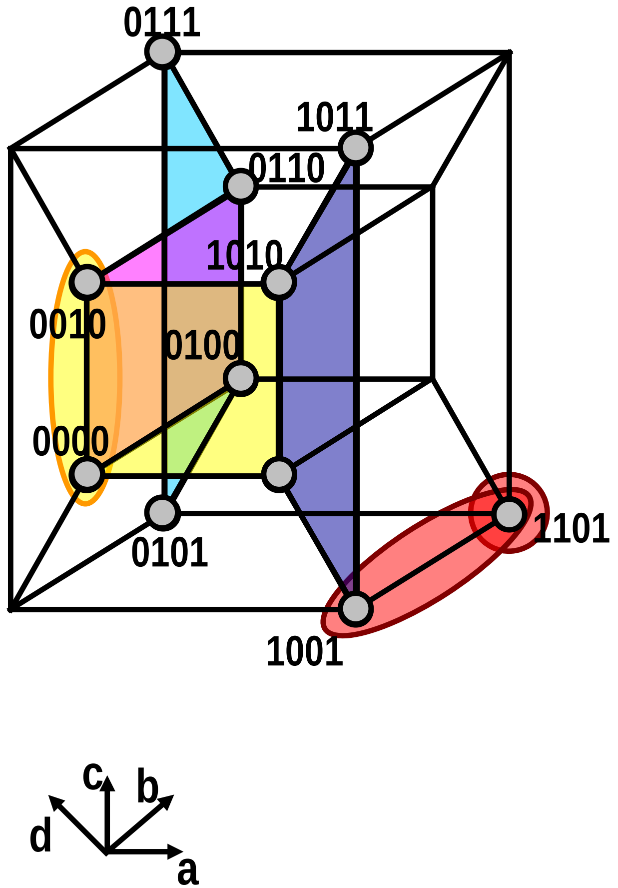
Example of expansion

- Expand 0000 to α = 0**0.
  - Drop 0100, 0010, 0110 from the cover.
- Expand 1000 to β = *0*0.
  - Drop 1010 from the cover.
- Expand 0101 to γ = 01**.
  - Drop 0111 from the cover.
- Expand 1001 to δ = 10**.
  - Drop 1011 from the cover.
- Expand 1101 to ε = 1*01.
- Cover is: {α,β,γ,δ,ε}.



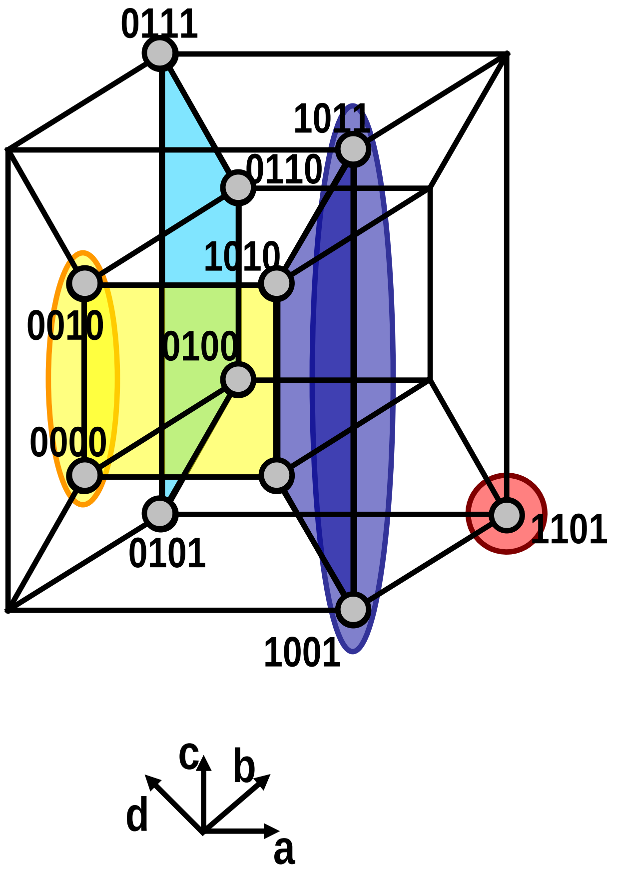
Example of reduction

- Reduce 0**0 to nothing.
- Reduce β = *0*0 to β' = 00*0.
- Reduce ε = 1*01 to ε' = 1101.
- Cover is: {β',γ,δ,ε'}.



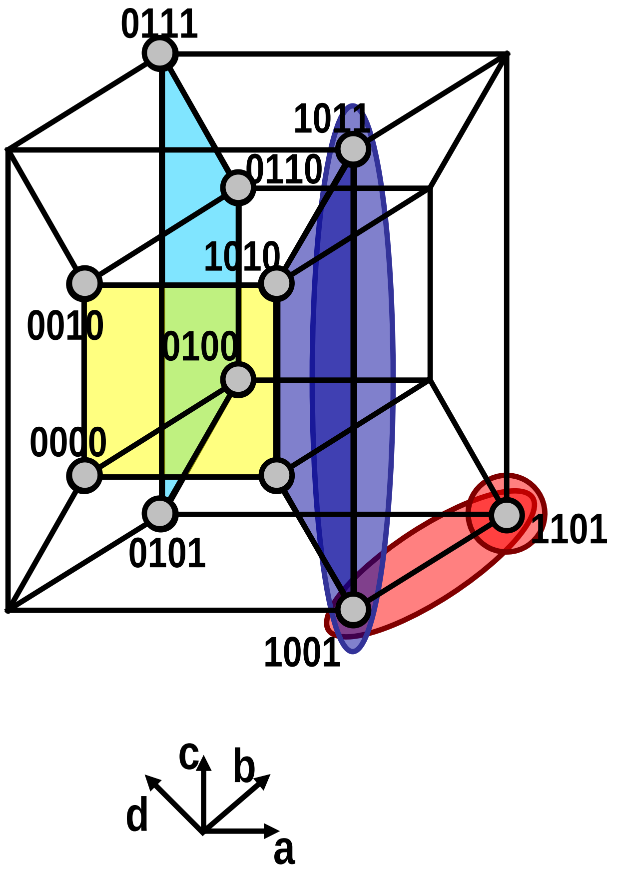
Example of reshape

- Reshape $\{\beta', \delta\}$ to: $\{\beta, \delta'\}$.
  - Where $\delta' = 10{*}1$.
- Cover is: $\{\beta, \gamma, \delta', \epsilon'\}$.



Example of second expansion

- Expand $\delta' = 10*1$ to $\delta = 10**$.
- Expand $\varepsilon' = 1101$ to $\varepsilon = 1*01$.

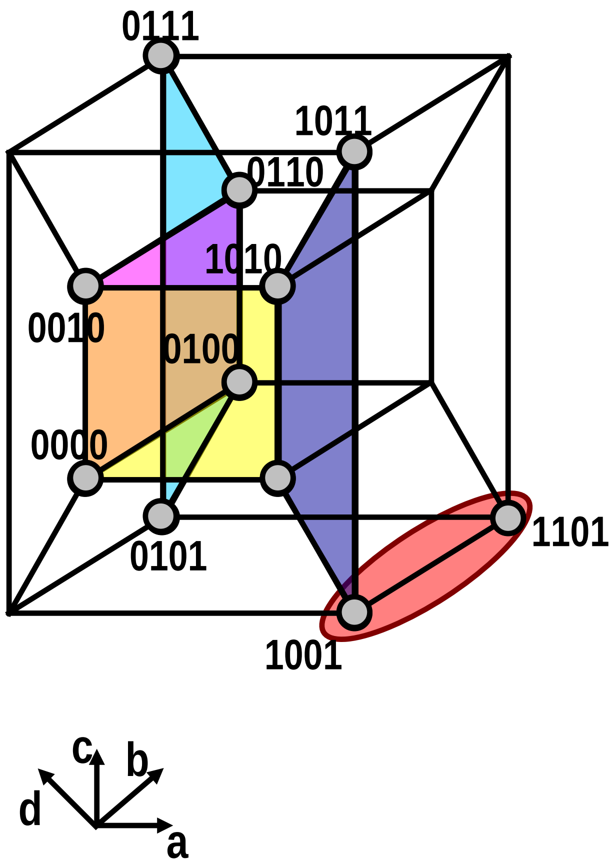


Example
Summary of the steps taken by MINI

- **Expansion:**
  - Cover: {α,β,γ,δ,ε}.
  - Prime, redundant, minimal w.r. to scc.
- **Reduction:**
  - α eliminated.
  - β = *0*0 reduced to β' = 00*0.
  - ε = 1*01 reduced to ε' = 1101.
  - Cover: {β',γ,δ,ε'}.
- **Reshape:**
  - {β', δ} reshaped to: {β, δ'} where δ' = 10*1.
- **Second expansion:**
  - Cover: {β,γ,δ,ε}.
  - Prime, irredundant.

Example
Summary of the steps taken by ESPRESSO

- **Expansion:**
  - Cover: {α,β,γ,δ,ε}.
  - Prime, redundant, minimal w.r.t. to scc.
- **Irredundant:**
  - Cover: {β,γ,δ,ε}.
  - Prime, irredundant.



Rough comparison of minimizers

- MINI
  - Iterate EXPAND, REDUCE, RESHAPE
- Espresso
  - Iterate EXPAND, IRREDUNDANT, REDUCE
- Espresso guarantees an irredundant cover
  - Because of the irredundant operator
- MINI may return irredundant covers, but can guarantee only minimality w.r.to single implicant containment

Expand
Naïve implementation

- For each implicant
  - For each care literal
    - Raise it to don't care if possible
  - Remove all implicants covered by expanded implicant
- Issues
  - Validity check of expansion
  - Order of expansion

Validity check

- **Espresso, MINI**
  - **Check intersection of expanded implicant with OFF-set**
  - **Requires complementation**
- **Presto**
  - **Check inclusion of expanded implicant in the union of the ON-set and DC-set**
  - **Reducible to recursive tautology check**

Ordering heuristics

- **Expand the cubes that are unlikely to be covered by other cubes**
- **Selection:**
  - **Compute vector of column sums**
  - ***Weight*: inner product of cube and vector**
  - **Sort implicants in ascending order of weight**
- **Rationale:**
  - **Low weight correlates to having few 1s in densely populated columns**

Example

- f = a'b'c' + ab'c' + a'bc' + a'b'c

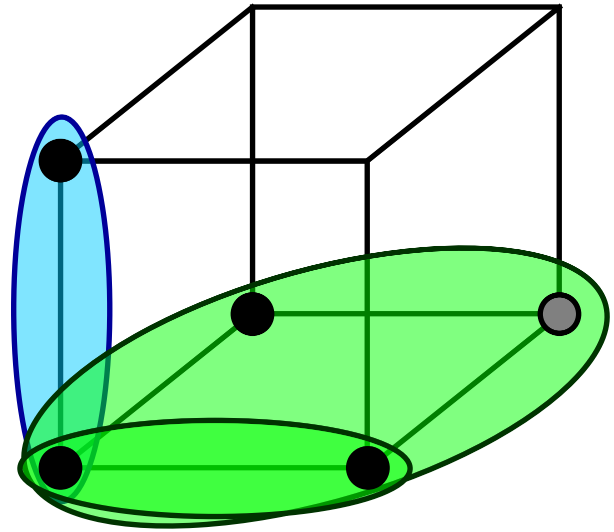
  DC-set = abc'

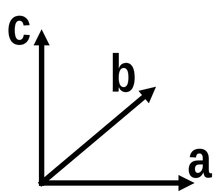
10	10	10
01	10	10
10	01	10
10	10	01

- **Ordering:**
  - **Vector: $[3\ 1\ 3\ 1\ 3\ 1]^T$**
  - **Weights: (9, 7, 7, 7)**
- **Select second implicant.**

Example (2)

α	10	10	10
β	01	10	10
γ	10	01	10
δ	10	10	01



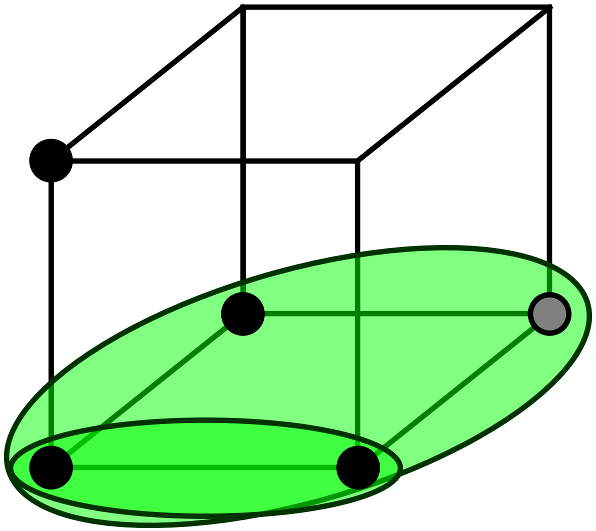


Example (3)

- **OFF-set:**

01	11	01
11	01	01

- **Expand 01 10 10:**
  - **11 10 10 valid.**
  - **11 11 10 valid.**
  - **11 11 11 invalid.**

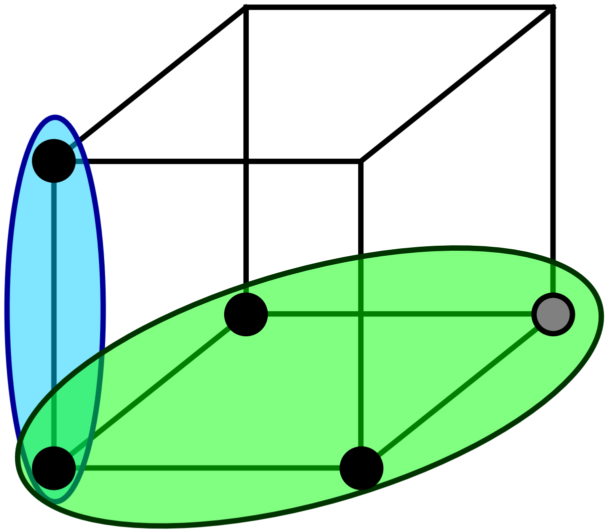


- **Update cover to:**

11	11	10
10	10	01

Example (4)

11	11	10
10	10	01



- Expand 10 10 01:
  - 11 10 01 invalid.
  - 10 11 01 invalid.
  - 10 10 11 valid.
- Expand cover:

11	11	10
10	10	11

Expand heuristics in ESPRESSO

- Special heuristic to choose the order of literals
- Rationale:
  - Raise literals to that expanded implicant
    - Covers a maximal set of cubes
    - Overlaps with a maximal set of cubes
    - The implicant is as large as possible
- Intuitive argument
  - Pair implicant to be expanded with other implicants, to check the fruitful directions for expansion

Expand in Espresso

- Compare implicant with OFF-set.
  - Determine possible and impossible directions of expansion
- Detection of feasibly covered implicants
  - If there is an implicant β whose supercube with α is feasible, expand α to that supercube and remove β
- Raise those literals of α to overlap a maximum number of implicants
  - It is likely that the uncovered part of those implicant is covered by some other expanded cube
- Find the largest prime implicant
  - Formulate a covering problem and solve it heuristically

Reduce

- **Sort implicants**
  - **Heuristics: sort by descending weight**
  - **Opposite to the heurstic sorting for expand**
- **Maximal reduction can be determine exactly**
- **Theorem:**
  - **Let α be in F and $Q = F \cup D - \{ \alpha \}$**
    Then, the maximally reduced cube is:
    $\acute{\alpha} = \alpha \cap \text{supercube}(Q'_{\alpha})$

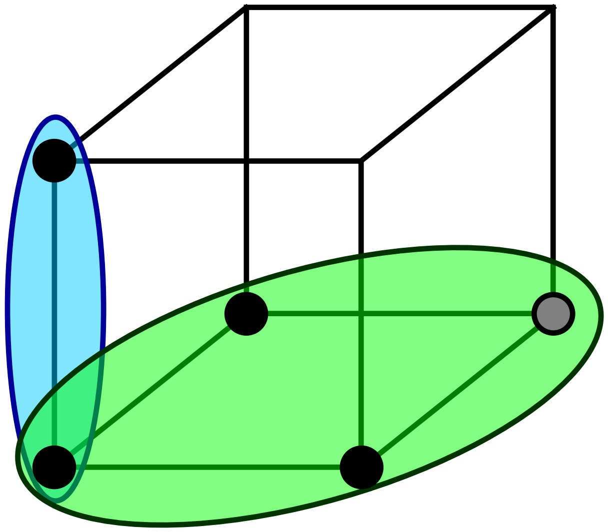
Example

- **Expand cover:**

  11 11 10
  10 10 11

- **Select first implicant:**
  - **Cannot be reduced.**
- **Select second implicant:**
  - **Reduced to 10 10 01**
- **Reduced cover:**

  11 11 10
  10 10 01



Irredundant cover

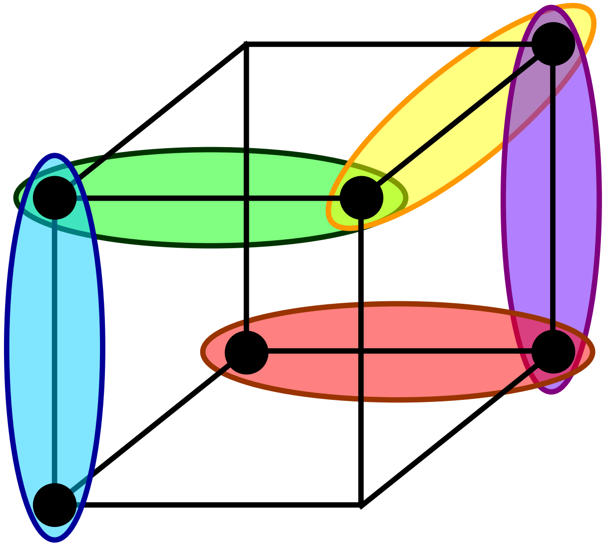
α 10 10 11

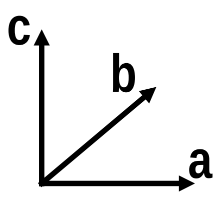
β 11 10 01

γ 01 11 01

δ 01 01 11

ε 11 01 10





Irredundant cover

- Relatively essential set E^r
  - Implicants covering some minterms of the function not covered by other implicants
  - Important remark: we do not know all the primes!
- Totally redundant set R^t
  - Implicants covered by the relatively essentials
- Partially redundant set R^p
  - Remaining implicants

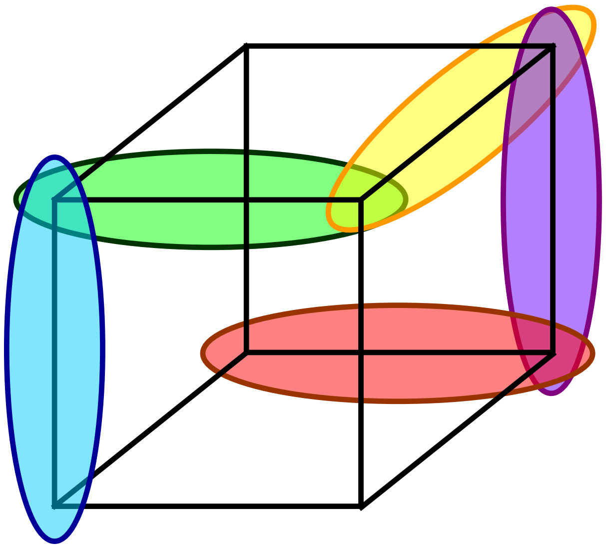
Irredundant cover

- Find a subset of R^p that, together with E^r covers the function
- Modification of the tautology algorithm
  - Each cube in R^p is covered by other cubes
  - Find mutual covering relations
- Reduces to a covering problem
  - Apply a heuristic algorithm.
  - Note that even by applying an exact algorithm, a minimum solution may not be found, because we do not have all primes.

Example

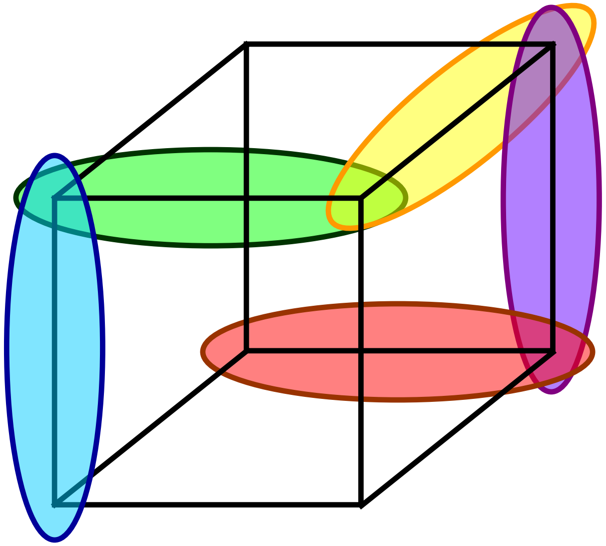
α	10 10 11
β	11 10 01
γ	01 11 01
δ	01 01 11
ε	11 01 10

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\beta, \gamma, \delta\}$



Example (2)

- **Covering relations:**
  - β is covered by $\{\alpha, \gamma\}$.
  - γ is covered by $\{\beta, \delta\}$.
  - δ is covered by $\{\gamma, \epsilon\}$.
- **Minimum cover:** $\gamma \cup E^r$



ESPRESSO algorithm in short

- Compute the complement
- Extract essentials
- Iterate
  - Expand, irredundant and reduce
- Cost functions:
  - Cover cardinality φ_1
  - Weighted sum of cube and literal count φ_2

ESPRESSO algorithm in detail

```
espresso(F,D) {
    R = complement(F U D);
    F = expand(F,R);
    F = irredundant(F,D);
    E = essentials(F,D);
    F = F – E;  D = D U E;
    repeat {
        φ2 = cost(F);
        repeat {
            φ1 = |F |;
            F = reduce(F,D);
            F = expand(F,R);
            F = irredundant(F,D);
        } until (|F | ≥ φ1);
        F = last_gasp(F,D,R);
    } until ( | F | ≥ φ1);
    F = F U E;  D = D – E;
    F = make_sparse(F,D,R);
}
```

Heuristic two-level minimization
Summary

- Heuristic minimization is iterative
- Few operators are applied to covers
- Underlying mechanism
  - Cube operation
  - Unate recursive mechanism
- Efficient algorithms