CAD for VLSI

Two-level Logic Optimization 2

Outline

- Data structures for logic optimization
- Data representation and encoding
- Operations on logic covers
- Application of the recursive paradigm
- Fundamental mechanisms used inside minimizers
- Heuristic two-level minimization
- The algorithms of ESPRESSO

Some More Background

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

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

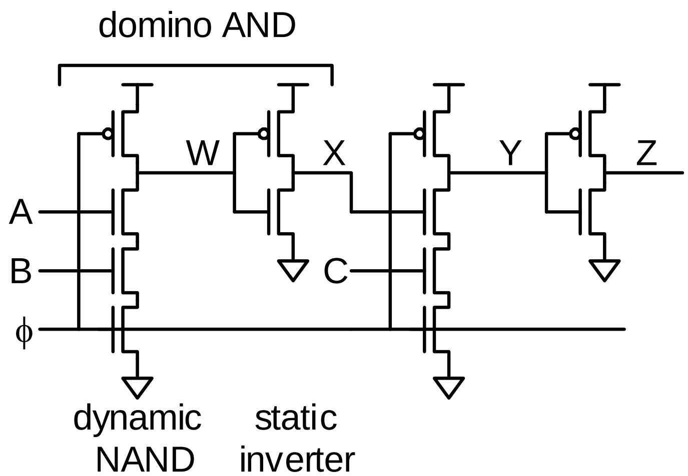
- Function $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$
- Positive unate in x_i when:
  - $f_{x_i} \supseteq f_{x_i'}$
  - $f = x_i f_{x_i} + x_i' f_{x_i'} = x_i f_{x_i} + f_{x_i'}$
- Negative unate in x_i when:
  - $f_{x_i} \subseteq f_{x_i'}$
  - $f = x_i f_{x_i} + x_i' f_{x_i'} = f_{x_i} + x_i' f_{x_i'}$
- A function is positive/negative unate
  when all its variables are positive/negative unate

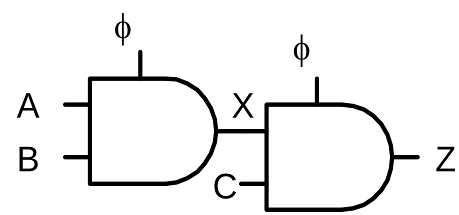
Binate

- Function f (x_1, x_2, …., x_i, …., x_n)
  - Mix of positive and negative phases
  - $(f_{x_i} \not\supseteq f_{x_i'}) \wedge (f_{x_i} \not\subseteq f_{x_i'})$

Recap Domino Circuit

- Domino only performs noninverting functions:
  - AND, OR but not NAND, NOR, or XOR
- $f_1(a, b, c) = ab + a'c$
  - Binate
- $f_2(a, b, c, d) = ab + dc$
  - Unate
  - where d = a' from PI



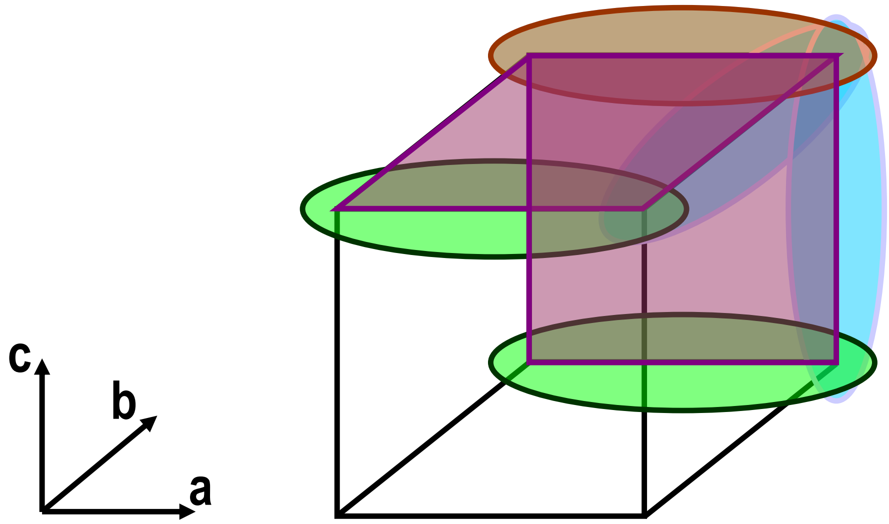


Operators

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

Example f = ab + bc + ac

- $f_a = b + c$; $f_{a'} = bc$
- 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

- Linear algebra
  - Basis of 3-dimensional vector space
    - (1, 0, 0) (0, 1, 0) (0, 0, 1) – linearly independent
    - (2, 0, 0) (0, 3, 0) (2, 1, 1) – linearly independent
    - (2, 0, 0) (0, 3, 0) (2, 1, 0) – linearly dependent
- Boolean algebra
  - Shannon expansion in terms of single variable
    - x, x'
    - xy, (xy)' ?

Generalized Expansion

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

    ϕ_i, i = 1, 2, … , k
  - $\sum_i^k \phi_i = 1, \ \phi_i \cdot \phi_j = 0$
- 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,...,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:

$\emptyset$	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
    - The smallest cube containing all cubes
- 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.t. β
  - Void when α does not intersect β
  - $a_1 + b_1'$ $a_2 + b_2'$ ... $a_n + b_n'$
- Cofactor of a set $C = \{\gamma_i\}$ w.r.t. β:
  - Set of cofactors of γ_i w.r.t. β

Example f = a'b' + ab

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

10 10 → a'b'

01 01 → ab

00 00

01 11

---

00 00 → 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

Outline

- Data structures for logic optimization
- Data representation and encoding
- Operations on logic covers
- Application of the recursive paradigm
- Fundamental mechanisms used inside minimizers
- Heuristic two-level minimization
- The algorithms of ESPRESSO

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:
  - Expand 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

Tautology Termination Rules

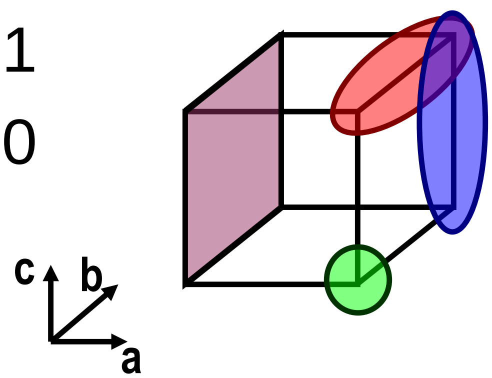
- TAUTOLOGY:
  - The cover matrix has a row of all 1s. (Tautology cube)
- NO TAUTOLOGY:
  - The cover has a column of 0s. (A variable never takes a value)
- TAUTOLOGY:
  - The cover depends on one variable, and there is no column of 0s in that field
- Decomposition rule:
  - When a cover is the union 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. a' → 10 11 11

  11 11 11 – Tautology
- Cofactor w.r.t. a → 01 11 11

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

11	01	11
11	11	01
11	10	10

- Select variable b
- Cofactor w.r.t. b' → 11 10 11

11	11	01
11	11	10

- No column of 0 - Tautology
- Cofactor w.r.t. b → 11 01 11

11 11 11 - Tautology

- Function is a *TAUTOLOGY*

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

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

11	11	01
11	11	10

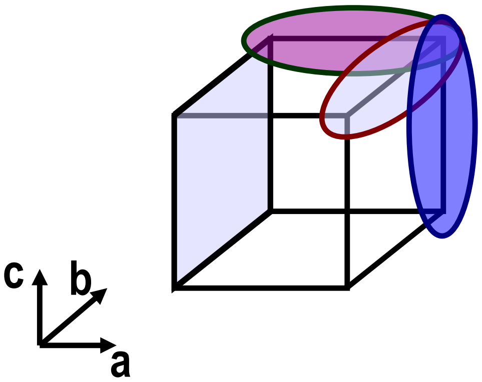
Containment

- 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

Complement Termination Rules

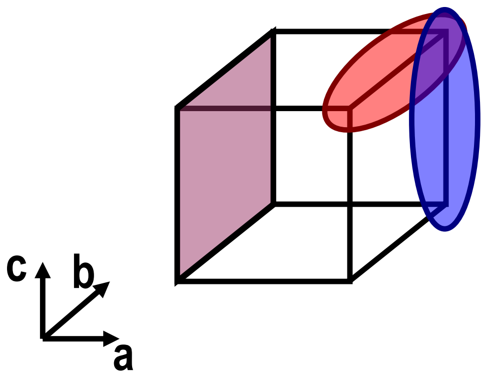
- The cover F is void
  - Hence its complement is the universal cube
- The cover F has a row of 1s
  - Hence F is a tautology and its complement is void
- The cover F consists of one implicant
  - Hence the complement is computed by DeMorgan's law
- All implicants of F depend on a single variable, and 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_i, then
    - $f = x_i\, f_{x_i} + x_i'\, f_{x_i'} = x_i\, f_{x_i} + f_{x_i'}$ $\quad (f_{x_i} \supseteq f_{x_i'})$
    - $f' = x_i\, f'_{x_i} + x_i'\, f'_{x_i'} = f'_{x_i} + x_i'\, f'_{x_i'}$ $\quad (f'_{x_i} \subseteq f'_{x_i'})$
  - If f is negative unate in x_i, then
    - $f = x_i\, f_{x_i} + x_i'\, f_{x_i'} = f_{x_i} + x_i'\, f_{x_i'}$ $\quad (f_{x_i} \subseteq f_{x_i'})$
    - $f' = x_i\, f'_{x_i} + x_i'\, f'_{x_i'} = x_i\, f'_{x_i} + f'_{x_i'}$ $\quad (f'_{x_i} \supseteq f'_{x_i'})$
- 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'

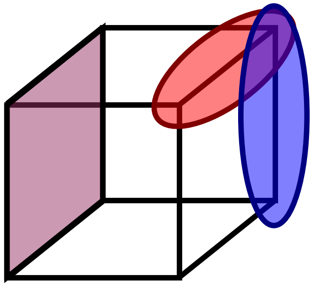
- Select binate variable a



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

11	01	11
11	11	01

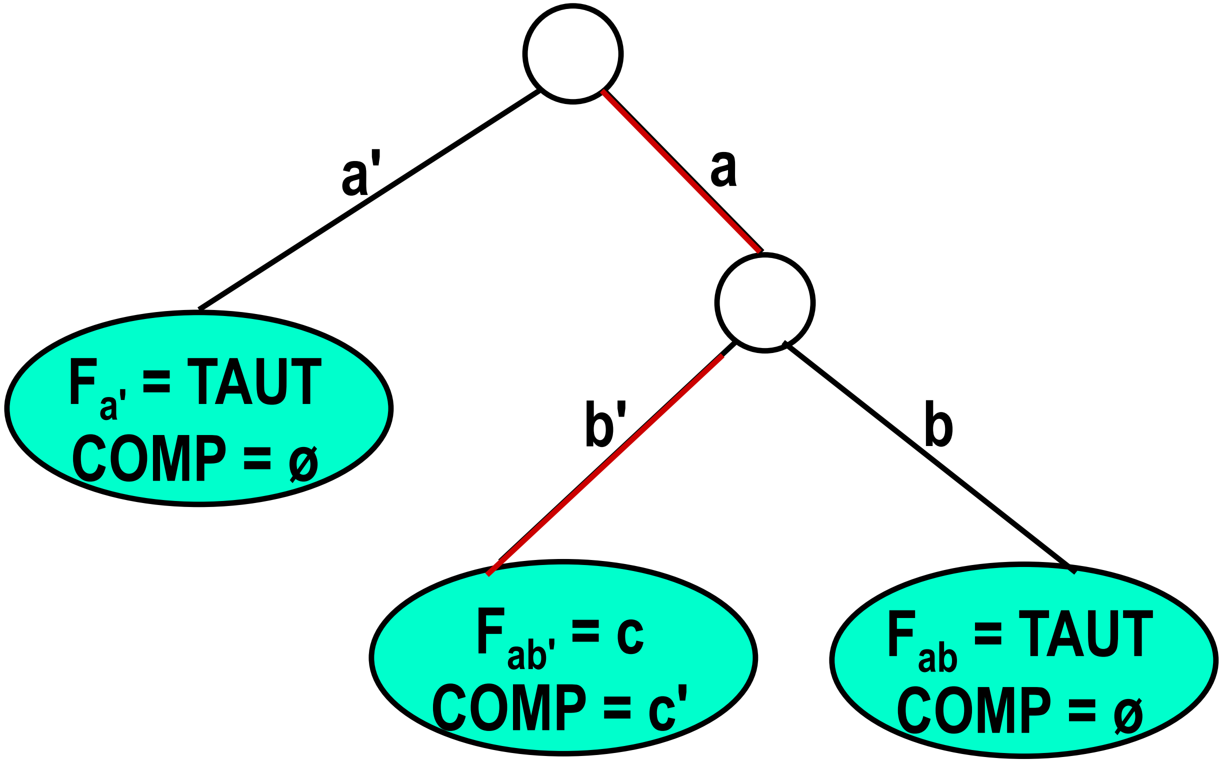
Example (2)



- 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

Outline

- Data structures for logic optimization
- Data representation and encoding
- Operations on logic covers
- Application of the recursive paradigm
- Fundamental mechanisms used inside minimizers
- 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

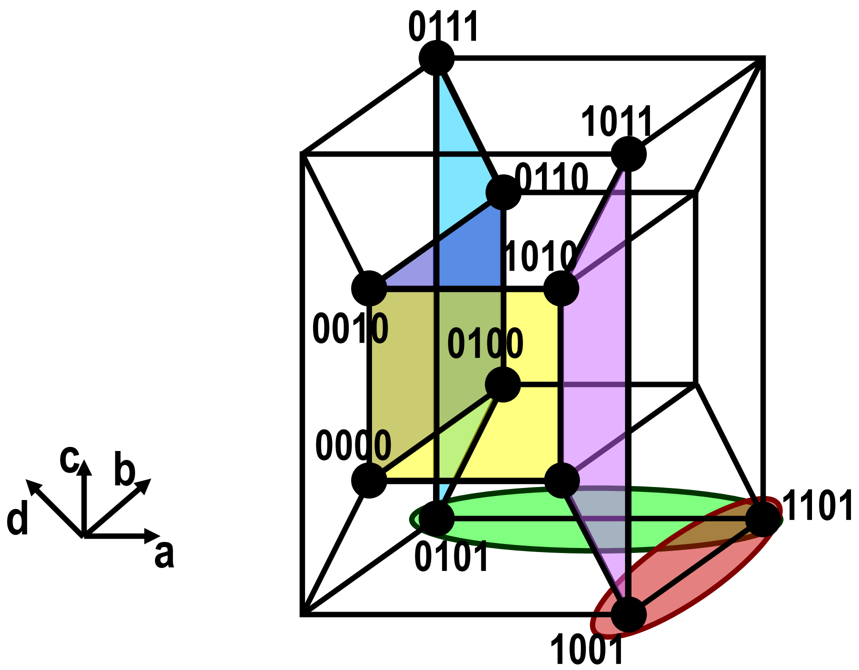
Example

- Initial cover
  - For simplicity, we use implicant table rather than 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

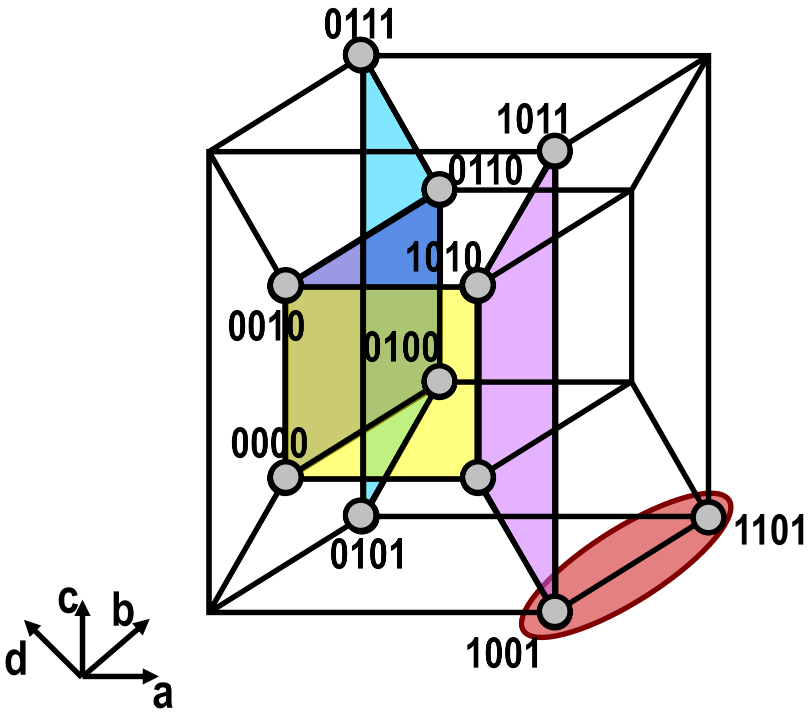
- Set of primes



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

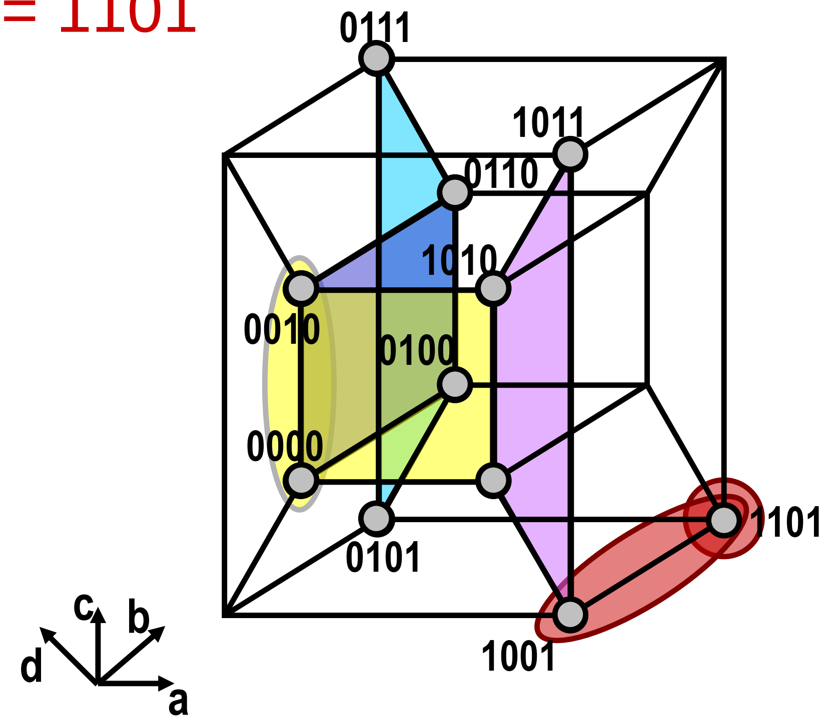
Example of Expansion

- 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}



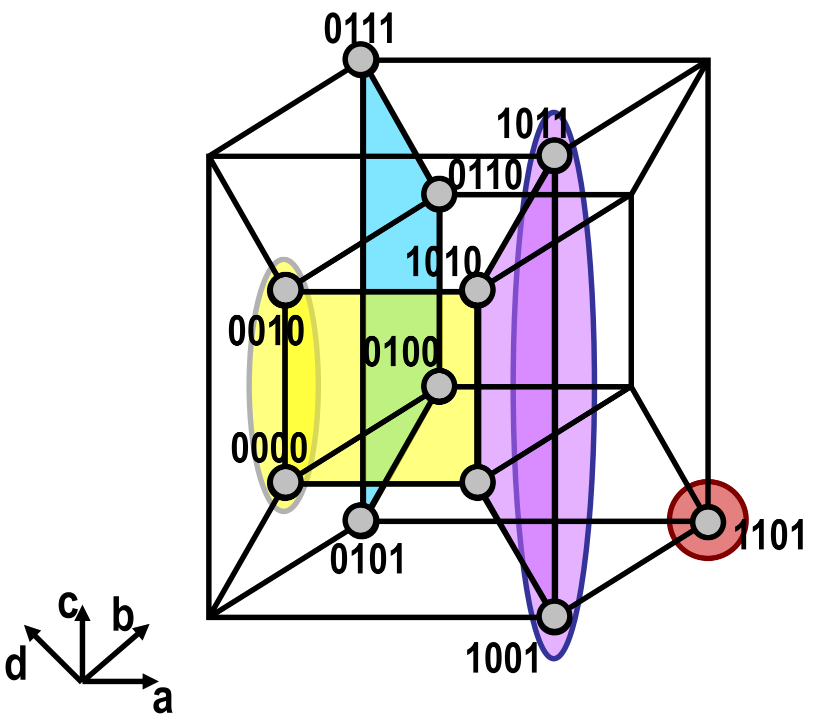
Example of Reduction

- Reduce α = 0**0 to nothing
- Reduce β = *0*0 to β' = 00*0
- Reduce ε = 1*01 to ε' = 1101
- Cover is: $\{\beta', \gamma, \delta, \varepsilon'\}$



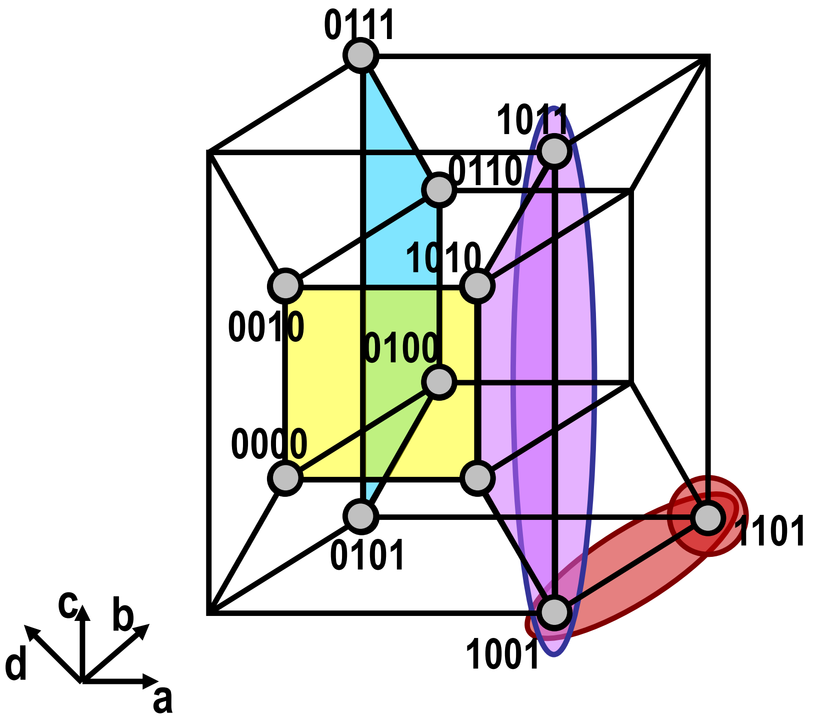
Example of Reshape

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



Example of Second Expansion

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

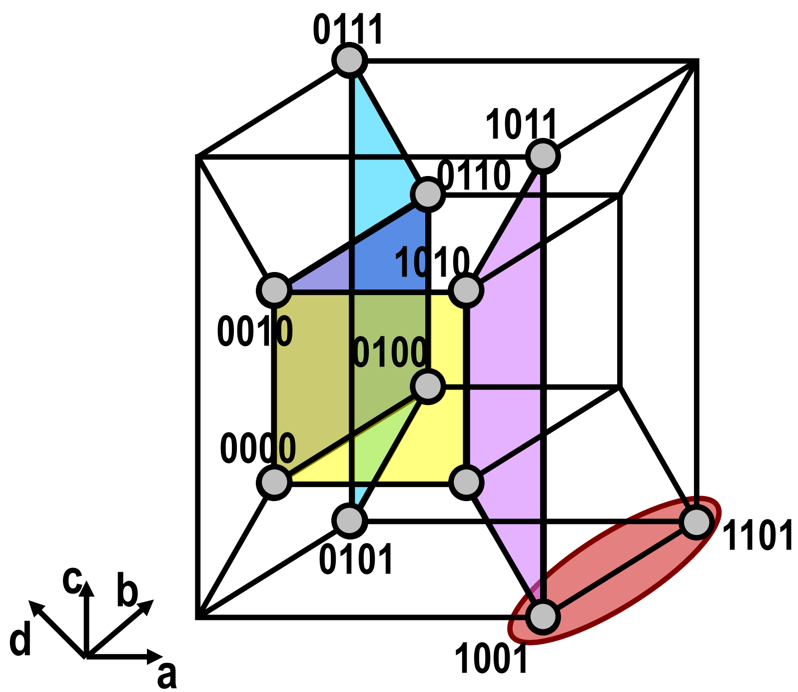


Example Summary of the Steps Taken by MINI

- Expansion:
  - Cover: $\{\alpha, \beta, \gamma, \delta, \varepsilon\}$
  - Prime, redundant, minimal w.r.t. scc
- Reduction:
  - α eliminated
  - β = *0*0 reduced to β' = 00*0
  - ε = 1*01 reduced to ε' = 1101
  - Cover: $\{\beta', \gamma, \delta, \varepsilon'\}$
- Reshape:
  - $\{\beta', \delta\}$ reshaped to: $\{\beta, \delta'\}$ where δ' = 10*1
- Second expansion:
  - Cover: $\{\beta, \gamma, \delta, \varepsilon\}$
  - Prime, irredundant

Example Summary of the Steps Taken by ESPRESSO

- Expansion:
  - Cover: $\{\alpha, \beta, \gamma, \delta, \varepsilon\}$
  - Prime, redundant, minimal w.r.t. scc
- Irredundant:
  - Cover: $\{\beta, \gamma, \delta, \varepsilon\}$
  - 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.t. 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

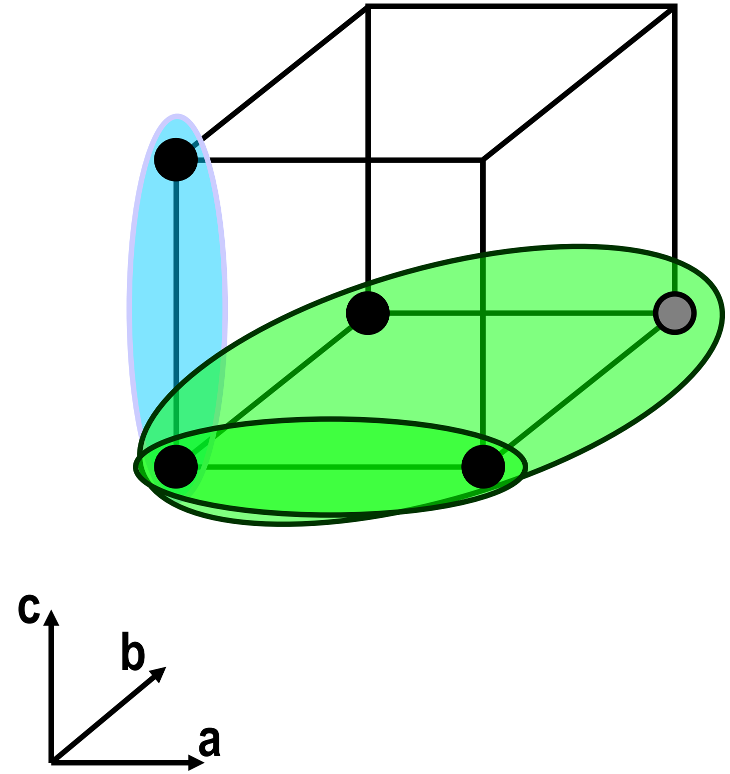
- f = a'b'c' + ab'c' + a'bc' + a'b'c
  DC-set = abc'

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



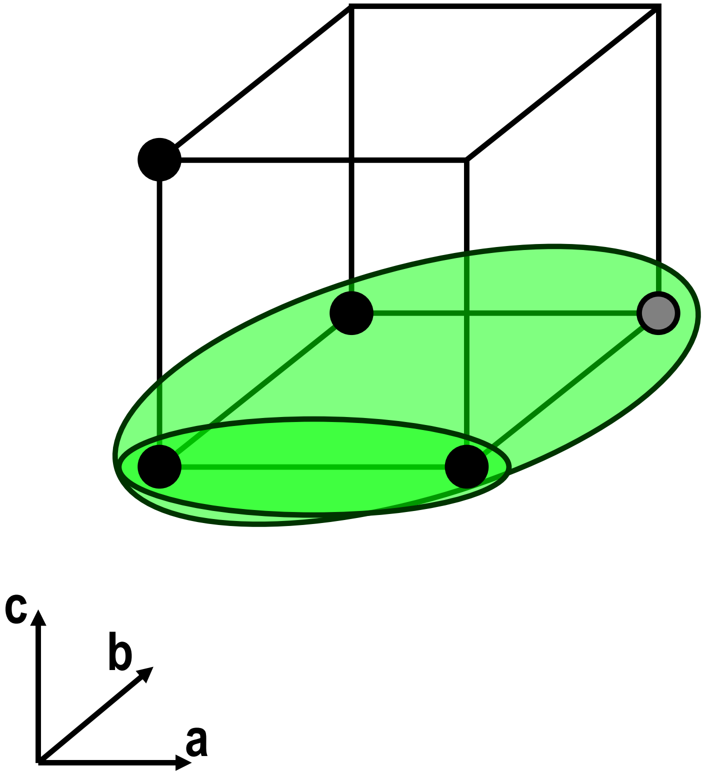
Example (3)

- 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



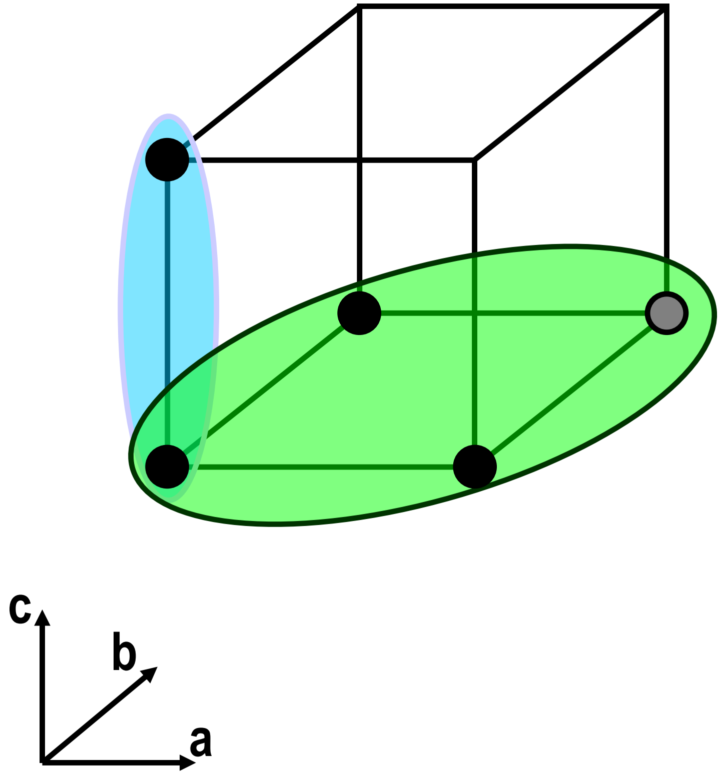
Example (4)

- OFF-set:

  01 11 01
  11 01 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 heuristic 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:
    $\acute{\alpha} = \alpha \cap \text{supercube}(Q'_{\alpha})$

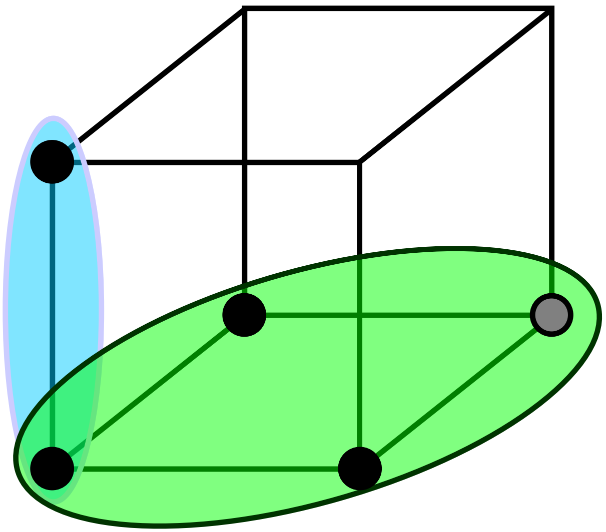
Example

- 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



Example

- Expand cover:

$\alpha \rightarrow$ **11 11 10**

$Q \rightarrow$ **10 10 11**

Q'

01 11 11 (a)

11 01 11 (b)

α 11 11 10

01 11 10

11 01 10

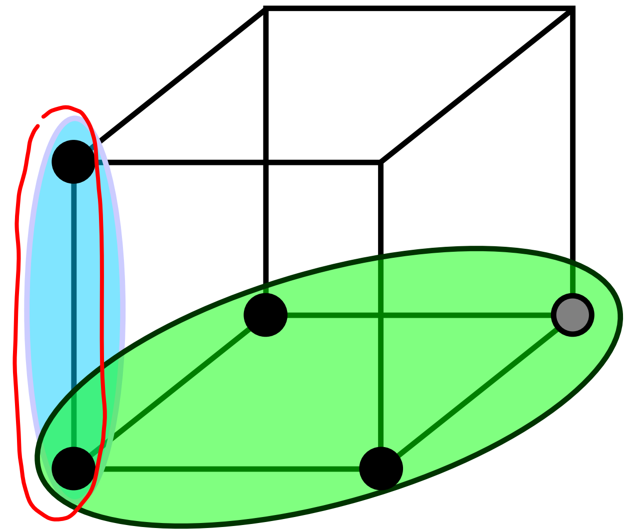
00 00 01

$Q'_\alpha \rightarrow$ 01 11 11

11 01 11

$\Rightarrow$ Supercube $\rightarrow$ 11 11 11

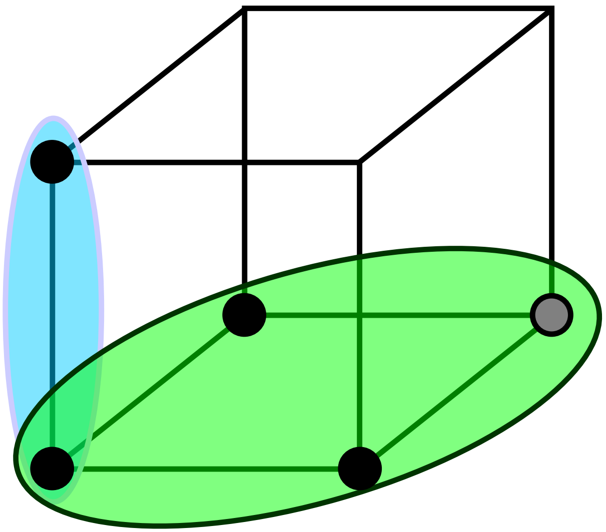
$Q'_\alpha \cdot \alpha$ = 11 11 10



Example

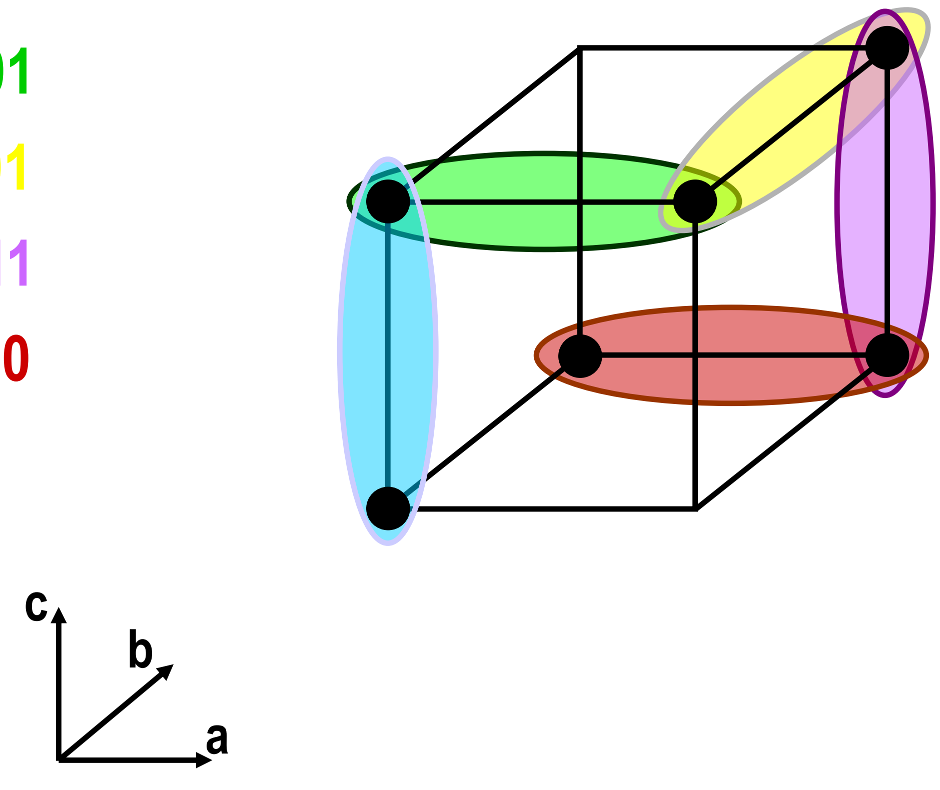
- Expand cover:

11	11	10
10	10	11



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

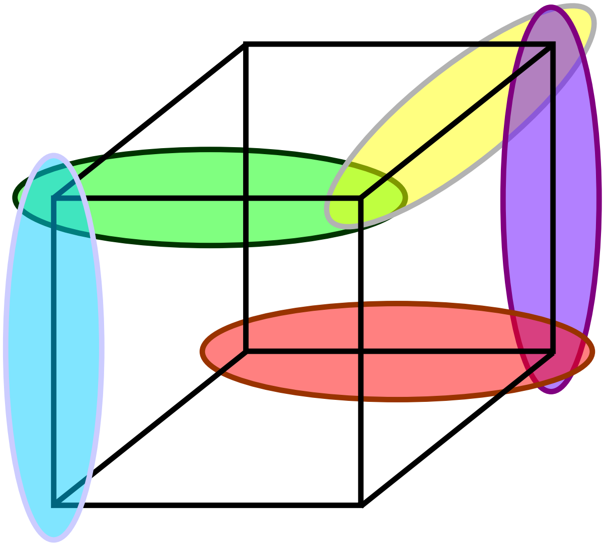
Irredundant Cover

- 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

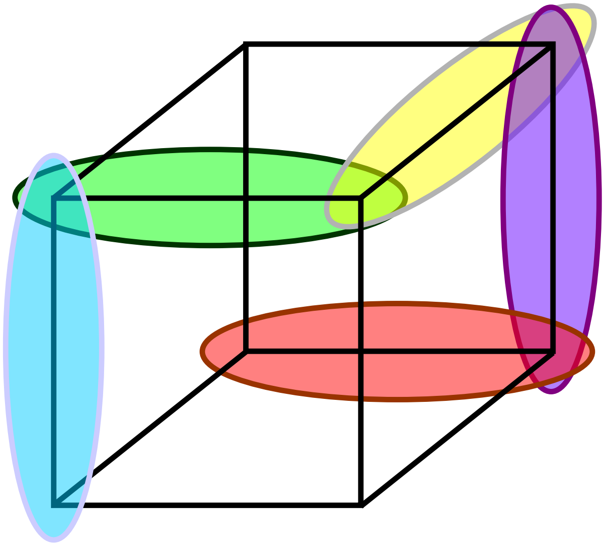
α	10	10	11
β	11	10	01
γ	01	11	01
δ	01	01	11
ε	11	01	10

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



Example (2)

- Covering relations:
  - β is covered by $\{\alpha, \gamma\}$
  - γ is covered by $\{\beta, \delta\}$
  - δ is covered by $\{\gamma, \varepsilon\}$
- Minimum cover: γ U 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 | < φ2);
    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