Heuristic Two-level Logic Optimization

Giovanni De Micheli
Integrated Systems Centre
EPF Lausanne

This presentation can be used for non-commercial purposes as long as this note and the copyright footers are not removed

© Giovanni De Micheli – All rights reserved
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 $x_i$
  $\Delta f_{x_i} = f (x_1, x_2, \ldots, 1, \ldots, x_n)$

◆ Cofactor of $f$ with respect to variable $x_i'$
  $\Delta f_{x_i'} = f (x_1, x_2, \ldots, 0, \ldots, x_n)$

◆ Boole’s expansion theorem:
  $\Delta f (x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i f_{x_i} + x_i' f_{x_i'}$

Also credited to Claude Shannon
Example

◆ Function: \( f = ab + bc + ac \)

◆ Cofactors:
  \( \Delta f_a = b + c \)
  \( \Delta f_a' = bc \)

◆ Expansion:
  \( \Delta f = a f_a + a' f_a' = a(b + c) + a' bc \)
Unateness

◆ Function \( f( x_1, x_2, \ldots, x_i, \ldots, x_n) \)

◆ Positive unate in \( x_i \) when:
  \[ f_{x_i} \geq f_{x_i}' \]

◆ Negative unate in \( x_i \) when:
  \[ f_{x_i} \leq f_{x_i}' \]

◆ 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** $x_i$:
  $$\frac{\partial f}{\partial x_i} \equiv f_{x_i} \oplus f_{x_i}'$$

- **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: $\phi_i, i = 1, 2, \ldots, k$

Then:

- $f = \sum_{i=1}^{k} \phi_i \cdot f_{\phi_i}$
- Where $f_{\phi_i}$ 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:

$\Delta ab \subseteq f_{\phi_1} \subseteq 1$

$\Delta a' bc + ab' c \subseteq f_{\phi_2} \subseteq ab + bc + ac$

◆ 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: \( \phi_i \), \( i=1,2,\ldots,k \)
  ▲ 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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ø</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>*</td>
<td>11</td>
</tr>
</tbody>
</table>
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

\[ f = a'd' + a'b + ab' + ac'd \]

\[
\begin{array}{cccc}
10 & 11 & 11 & 10 \\
10 & 01 & 11 & 11 \\
01 & 10 & 11 & 11 \\
01 & 11 & 10 & 01 \\
\end{array}
\]
Cofactor computation

- Cofactor of $\alpha$ w.r. to $\beta$
  - Void when $\alpha$ does not intersect $\beta$
  - $a_1 + b_1', a_2 + b_2', \ldots, a_n + b_n'$

- Cofactor of a set $C = \{\gamma_i\}$ w.r. to $\beta$:
  - Set of cofactors of $\gamma_i$ w.r. to $\beta$
**Example** \( f = a' b' + ab \)

- **Cofactor w.r. to** \( \begin{array}{cc} 01 & 11 \\ \end{array} \)
  - First row – void
    - \( \begin{array}{cc} 01 & 01 \\ \end{array} \)
  - Second row – \( \begin{array}{cc} 11 & 01 \\ \end{array} \)
    - Cofactor \( f_a = b \)

\[
\begin{array}{cc}
10 & 10 \\
01 & 01 \\
00 & 00 \\
01 & 11 \\
00 & 00 \\
10 & 00 \\
11 & 01 \\
\end{array}
\]

\[\text{void}\]

---

(c) Giovanni De Micheli
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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>001</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>001</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>110</td>
</tr>
</tbody>
</table>
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:
  ▲ 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
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. to \( a' \) is
  \[ \begin{array}{ccc}
  11 & 11 & 11 \\
  \end{array} \]

- Cofactor w.r. to \( a \) is:
  \[ \begin{array}{c|ccc}
  & 01 & 11 & 01 \\
  11 & 11 & 11 & 10 \\
  11 & 11 & 01 & 10 \\
  11 & 10 & 10 & 10 \\
  \end{array} \]
### Example (2)

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

- Select variable $b$

- Cofactor w.r. to $b'$ is

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

- No column of 0 - Tautology

- Cofactor w.r. to $b$:

  Has row of 1s

- Function is a **TAUTOLOGY**
Containment

◆ Theorem:

▲ A cover $F$ contains an implicant $\alpha$ if and only if $F_\alpha$ is a tautology

◆ Consequence:

▲ Containment can be verified by the tautology algorithm
Check covering of $bc : 11 \ 01 \ 01$.

Take the cofactor:

$$
\begin{array}{ccc}
01 & 11 & 11 \\
01 & 11 & 11 \\
10 & 11 & 11 \\
\end{array}
$$

Tautology – $bc$ is contained by $f$. 

\[ f = ab + ac + a' \]
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
  \[ \nabla f' = f'_x + x' f'_x. \]
  ▲ If $f$ is negative unate in $x$, then
  \[ \nabla f' = x f'_x + f'_x. \]

◆ 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:
    
    $\begin{bmatrix}
    11 & 01 & 11 \\
    11 & 11 & 01 \\
    11 & 11 & 01 
    \end{bmatrix}$
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:

- $F_{a'} = \text{TAUT}$
  - $\text{COMP} = \emptyset$

- $F_{ab'} = c$
  - $\text{COMP} = c'$

- $F_{ab} = \text{TAUT}$
  - $\text{COMP} = \emptyset$

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)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Giovanni De Micheli
Set of all primes

\[ \begin{array}{cccc}
\alpha & 0 & * & * & 0 & 1 \\
\beta & * & 0 & * & 0 & 1 \\
\gamma & 0 & 1 & * & * & 1 \\
\delta & 1 & 0 & * & * & 1 \\
\epsilon & 1 & * & 0 & 1 & 1 \\
\zeta & * & 1 & 0 & 1 & 1 \\
\end{array} \]
Example of expansion

◆ Expand 0000 to $\alpha = 0^{**}0$.
  ▲ Drop 0100, 0010, 0110 from the cover.

◆ Expand 1000 to $\beta = *0*0$.
  ▲ Drop 1010 from the cover.

◆ Expand 0101 to $\gamma = 01^{**}$.
  ▲ Drop 0111 from the cover.

◆ Expand 1001 to $\delta = 10^{**}$.
  ▲ Drop 1011 from the cover.

◆ Expand 1101 to $\epsilon = 1*01$.

◆ Cover is: $\{\alpha, \beta, \gamma, \delta, \epsilon\}$. 
Example of reduction

- Reduce $0^{**}0$ to nothing.
- Reduce $\beta = *0*0$ to $\beta' = 00*0$.
- Reduce $\epsilon = 1*01$ to $\epsilon' = 1101$.
- Cover is: $\{\beta', \gamma, \delta, \epsilon'\}$. 
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. to scc.

◆ Reduction:
  ▲ \alpha eliminated.
  ▲ \beta = *0*0 reduced to \beta' = 00*0.
  ▲ \varepsilon = 1*01 reduced to \varepsilon' = 1101.
  ▲ Cover: \{\beta', \gamma, \delta, \varepsilon'\}.

◆ Reshape:
  ▲ \{\beta', \delta\} reshaped to: \{\beta, \delta'\} where \delta' = 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. to 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.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'

\[
\begin{array}{ccc}
10 & 10 & 10 \\
01 & 10 & 10 \\
10 & 01 & 10 \\
10 & 10 & 01 \\
\end{array}
\]

◆ Ordering:

▲ Vector: [3 1 3 1 3 1]^T

▲ Weights: (9, 7, 7, 7)

◆ Select second implicant.
Example (2)

\[\begin{align*}
\alpha & : 10\ 10\ 10 \\
\beta & : 01\ 10\ 10 \\
\gamma & : 10\ 01\ 10 \\
\delta & : 10\ 10\ 01 \\
\text{DC} & : 01\ 01\ 10
\end{align*}\]
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
```

(c) Giovanni De Micheli
Example (4)

\[
\begin{array}{ccc}
11 & 11 & 10 \\
10 & 10 & 01
\end{array}
\]

◆ Expand 10 10 01:
  ▲ 11 10 01 invalid.
  ▲ 10 11 01 invalid.
  ▲ 10 10 11 valid.

◆ Expanded cover:

\[
\begin{array}{ccc}
11 & 11 & 10 \\
10 & 10 & 11
\end{array}
\]
Expand heuristics in ESPRESSO

◆ Special heuristic to choose the order of literals

◆ Rationale:

▲ Raise literals so that the 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 $\beta$ whose supercube with $\alpha$ is feasible, expand $\alpha$ to that supercube and remove $\beta$
◆ Raise those literals of $\alpha$ 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 determined exactly
◆ Theorem:
  ▲ Let \( \alpha \) be in \( F \) and \( Q = F \cup D - \{ \alpha \} \)
  Then, the maximally reduced cube is:
  \( \hat{\alpha} = \alpha \cap \text{supercube} \left( Q' \alpha \right) \)
Example

◆ Expand cover:

\[
\begin{array}{ccc}
11 & 11 & 10 \\
10 & 10 & 11 \\
10 & 10 & 01 \\
11 & 11 & 10 \\
\end{array}
\]

◆ Select first implicant:

▲ Cannot be reduced.

◆ Select second implicant:

▲ Reduced to 10 10 01

◆ Reduced cover:

\[
\begin{array}{ccc}
11 & 11 & 10 \\
10 & 10 & 01 \\
\end{array}
\]
Irredundant cover

\[\begin{align*}
\alpha & : 10 10 11 \\
\beta & : 11 10 01 \\
\gamma & : 01 11 01 \\
\delta & : 01 01 11 \\
\epsilon & : 11 01 10
\end{align*}\]
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 $\mathbb{R}^p$ that, together with $E^r$ covers the function

◆ Modification of the tautology algorithm
  ▲ Each cube in $\mathbb{R}^p$ 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

- $E^r = \{\alpha, \varepsilon\}$
- $R^t = \emptyset$
- $R^p = \{\beta, \gamma, \delta\}$
Example (2)

◆ Covering relations:
   ▲ β is covered by {α, γ}.
   ▲ γ is covered by {β, δ}.
   ▲ δ is covered by {γ, ε}.

◆ Minimum cover: γ U Er
ESPRESSO algorithm in short

- Compute the complement
- Extract essentials
- Iterate
  - Expand, irredundant and reduce
- Cost functions:
  - Cover cardinality $\varphi_1$
  - Weighted sum of cube and literal count $\varphi_2$
ESPRESSO algorithm in detail

```plaintext
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 {
        φ_2 = cost(F);
        repeat {
            φ_1 = |F |;
            F = reduce(F,D);
            F = expand(F,R);
            F = irredundant(F,D);
        } until (|F | ≥ φ_1);
        F = last_gasp(F,D,R);
    } until (cost( F ) ≥ φ_2);
    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