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** \( x_i \)
  - \( f_{xi} = f (x_1, x_2, \ldots, 1, \ldots, x_n) \)

- **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:**
  - \( f (x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i f_{x_i} + x_i' f_{x_i'} \)
  - 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:
  - $f_{x_i} \geq f_{x_i'}$

◆ *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 $f \left( x_1, x_2, \ldots, x_i, \ldots, x_n \right)$

◆ 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 \( \frac{\partial f}{\partial a} = f_a \oplus f_{a'} = b'c + bc' \)
- 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 \textit{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 \( f\phi_2 = a'bc + ab'c \).

\[
\begin{align*}
  f &= \phi_1 f\phi_1 + \phi_2 f\phi_2 \\
  &= ab1 + (a' + b')(a'bc + ab'c) \\
  &= ab + bc + ac
\end{align*}
\]
Generalized expansion theorem

Given:

- Two function $f$ and $g$.
- Orthonormal set of functions: $\phi_i$, $i=1,2,...,k$
- Boolean operator $\odot$

Then:

$\star f \odot g = \sum_{i}^{k} \phi_i \cdot (f \phi_i \odot g \phi_i)$

Corollary:

$\star f \odot g = x_i \cdot (f_{x_i} \odot g_{x_i}) + x_i' \cdot (f_{x_i'} \odot g_{x_i'})$
Matrix representation of logic covers

◆ Representations used by logic minimizers

◆ Different formats
  ♦ Usually one row per implicant

◆ Symbols:
  ♦ 0, 1, *, ...

◆ Encoding:

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

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

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>01</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>
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** \(01 \quad 11\)  
  - First row – void  
  - Second row – \(11 \quad 01\)

- **Cofactor** \(f_a = b\)

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

\(\text{void}\)
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

(c) Giovanni De Micheli
Example

◆ 2-input, 3-output function:

- $f_1 = a'b' + ab$
- $f_2 = ab$
- $f_3 = ab' + a'b$

◆ Mvi representation:

```
10 10 100
01 10 001
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:
  • 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** \( 11 \ 11 \ 11 \) – Tautology.

- **Cofactor w.r. to** \( a \) **is:**

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

(c) Giovanni De Micheli
Example (2)

|| 01 | 11 |
---|----|----|
11 | 11 | 01 |
11 | 10 | 10 |

◆ Select variable \( b \).

◆ Cofactor w.r. to \( b' \) is

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

◆ No column of 0 - Tautology

<table>
<thead>
<tr>
<th>11</th>
<th>00</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

◆ Cofactor w.r. to \( b \) is:

11 11 11

◆ 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
Example
\[ f = ab + ac + a' \]

◆ Check covering of \(bc : 11\ 01\ 01\).

◆ Take the cofactor:

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

◆ 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:

\[
\begin{array}{ccc}
11 & 01 & 11 \\
11 & 11 & 01 \\
\end{array}
\]
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 \( \text{Cube}(b') = 11 \ 10 \ 11 \) yields \( 11 \ 10 \ 10 \)
  - \( 11 \ 10 \ 10 \) intersected with \( \text{Cube}(a) = 01 \ 11 \ 11 \) yields \( 01 \ 10 \ 10 \)

◆ Complement: \( F' = 01 \ 10 \ 10 \)
Example (3)

Recursive search:

Complement: \(a \ b'c'\)
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

(c) Giovanni De Micheli
Example

Set of primes

\[
\begin{array}{c|cccc|c}
\alpha & 0 & * & * & 0 & 1 \\
\beta & * & 0 & * & 0 & 1 \\
\gamma & 0 & 1 & * & * & 1 \\
\delta & 1 & 0 & * & * & 1 \\
\epsilon & 1 & * & 0 & 1 & 1 \\
\zeta & * & 1 & 0 & 1 & 1 \\
\end{array}
\]
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 $\varepsilon = 1^{*}01$.
◆ Cover is: $\{\alpha, \beta, \gamma, \delta, \varepsilon\}$.
Example of reduction

- Reduce 0**0 to nothing.
- Reduce $\beta = *0*0$ to $\beta' = 00*0$.
- Reduce $\varepsilon = 1*01$ to $\varepsilon' = 1101$.
- 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, \epsilon\}.
- Prime, redundant, minimal w.r. to scc.

Reduction:
- \alpha eliminated.
- \beta = \ast 0 \ast 0 reduced to \beta' = 00 \ast 0.
- \epsilon = 1 \ast 01 reduced to \epsilon' = 1101.
- Cover: \{\beta', \gamma, \delta, \epsilon\}.

Reshape:
- \{\beta', \delta\} reshaped to: \{\beta, \delta'\} where \delta' = 10 \ast 1.

Second expansion:
- Cover: \{\beta, \gamma, \delta, \epsilon\}.
- Prime, irredundant.
Example

Summary of the steps taken by ESPRESSO

Expansion:

- Cover: \{α,β,γ,δ,ε\}.
- Prime, redundant, minimal w.r. 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'

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

Ordering:
- Vector: \([3 \ 1 \ 3 \ 1 \ 3 \ 1]^T\)
- Weights: \((9, 7, 7, 7)\)

Select second implicant.
Example (2)

\begin{align*}
\alpha & \quad 10 \ 10 \ 10 \\
\beta & \quad 01 \ 10 \ 10 \\
\gamma & \quad 10 \ 01 \ 10 \\
\delta & \quad 10 \ 10 \ 01 \\
\end{align*}
Example (3)

- **OFF-set:**
  
  \[
  \begin{array}{ccc}
  01 & 11 & 01 \\
  11 & 01 & 01 \\
  \end{array}
  \]

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

- **Update cover to:**
  
  \[
  \begin{array}{ccc}
  11 & 11 & 10 \\
  10 & 10 & 01 \\
  \end{array}
  \]
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.

- Expand 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 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 $\beta$ 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 determine exactly

◆ Theorem:
  ♦ Let \( \alpha \) be in \( F \) and \( Q = F \cup D - \{ \alpha \} \)
  Then, the maximally reduced cube is:
  \( \hat{\alpha} = \alpha \cap \text{supercube} (Q'_\alpha) \)
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

\[\alpha \begin{array}{ccc} 10 & 10 & 11 \\ \beta & 11 & 10 & 01 \\ \gamma & 01 & 11 & 01 \\ \delta & 01 & 01 & 11 \\ \epsilon & 11 & 01 & 10 \end{array}\]
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

<table>
<thead>
<tr>
<th></th>
<th>(\alpha)</th>
<th>(\beta)</th>
<th>(\gamma)</th>
<th>(\delta)</th>
<th>(\varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E^r)</td>
<td>({\alpha, \varepsilon})</td>
<td>(\emptyset)</td>
<td>({\beta, \gamma, \delta})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (2)

Covering relations:

- $\beta$ is covered by $\{\alpha, \gamma\}$.
- $\gamma$ is covered by $\{\beta, \delta\}$.
- $\delta$ is covered by $\{\gamma, \varepsilon\}$.

Minimum cover: $\gamma \cup E^r$
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 {
        \( \phi_2 = \text{cost}(F); \)
        repeat {
            \( \phi_1 = |F|; \)
            F = reduce(F,D);
            F = expand(F,R);
            F = irredundant(F,D);
        } until (|F| \geq \phi_1);
        F = \text{last_gasp}(F,D,R);
    } until (|F| \geq \phi_1);
    F = F U E;  D = D - E;
    F = \text{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