ESPRESSO Logic Minimizer

ESPRESSO is a fast, efficient heuristic two-level logic minimizer. It makes use of

- Shannon expansion
- Unate recursive paradigm

\[ f = x f_x + \overline{x} f_{\overline{x}} \]  
(simplified for unate functions)

- Choose the splitting variable
- Operate on two cofactors
- Merge the results

\[ \text{operate}(f) = \text{merge} \{ x \cdot \text{operate}(f_x), \overline{x} \cdot \text{operate}(f_{\overline{x}}) \} \]

Operations:
- simplification, complement, tautology check, etc.
Matrix representation

$M(f)$ represents cover $F$ of function $f$:

$$M_{ij}(f) = \begin{cases} 
1 & \text{if variable } x_j \text{ appears in cube } c_i \\
0 & \text{if variable } \overline{x}_j \text{ appears in cube } c_i \\
2 & \text{(don't care) otherwise} 
\end{cases}$$

Example:

$$f = \overline{abc}d + ace + \overline{abcde} + bce$$

$$M(f) = \begin{bmatrix} 
1 & 2 & 1 & 2 & 1 & 0 \\
2 & 1 & 2 & 1 & 0 & 1 \\
1 & 0 & 1 & 2 & 0 & 0 \\
2 & 1 & 1 & 2 & 0 & 0 
\end{bmatrix}$$

Note: only input part of the cubes are shown.

Cofactoring example

Given function $f$ and its matrix $M$:

$$f = \overline{abc}d + ace + \overline{abcde} + bce$$

$$M(f) = \begin{bmatrix} 
0 & 1 & 2 & 1 & 2 \\
1 & 2 & 1 & 2 & 1 \\
0 & 0 & 1 & 1 & 0 \\
2 & 1 & 1 & 2 & 0 
\end{bmatrix}$$

Cofactors w.r. to variable $b$

$$f_b = \overline{ad} + ace + \overline{ce}$$

$$M(f_b) = \begin{bmatrix} 
0 & 2 & 2 & 1 & 2 \\
1 & 2 & 1 & 2 & 1 \\
2 & 2 & 1 & 2 & 0 
\end{bmatrix}$$

$$f_b = ace + \overline{a}cde$$

$$M(f_b) = \begin{bmatrix} 
1 & 2 & 1 & 2 & 1 \\
0 & 2 & 1 & 1 & 0 
\end{bmatrix}$$
Choice of splitting variable

Best splitting variable $x$: most binate (non-unate) variable

- covers $F_x$ and $F_{\bar{x}}$ become unate after a minimum number of splittings
- the total number of cubes in $F_x$ and $F_{\bar{x}}$ is minimum

Most binate variable is the one with maximum number of 0’s and 1’s in $M(f)$.

Detecting unateness & tautology

- Cover $F$ is unate if each column of $M(f)$ is
  - void of 0’s (pos. unate), or
  - void of 1’s (neg. unate)

- $F$ is the tautology if $M(f)$ has a row of all 2’s (don’t cares).
**Recursive tautology**

Follow the recursive paradigm, splitting at the most binate variable $x$

$$F = xF_x + \overline{x}F_{\overline{x}}$$

**Termination & simplification rules:**

- The cover $F$ is void:
  $$F = \emptyset$$

- $F$ has a row of all 2's (dc's): tautology
  $$F \equiv 1$$

- $F$ depends on only one variable $x$ which has both 0 and 1.

- When $F = F_1 \cup F_2$, with disjoint support, check tautology in $F_1$, $F_2$ separately.

If cover is positive (negative) unite at $x$, use:

$$f = xF_x + \overline{x}F_{\overline{x}} \quad \text{(or} \quad F = F_x + \overline{x}F_{\overline{x}})$$

---

**Tautology example**

Recursive splitting + evaluation

$$F = ab + ac + a' + ab'c'$$

![Tautology example diagram]
Recursive complementation

Follow the recursive paradigm, splitting at the most binate variable $x$ (to get unate cofactors)

$$F = xF_x + x\overline{F_x}$$

Termination & simplification rules:

- The cover $F$ is void: $F \equiv 1$.
- $F$ has a row of all 2's (dc's): $F \equiv 1$
  $$\overline{F} = 0$$
- $F$ has single row with one 0 or 1 at $x$:
  $$\overline{F} = x \text{ or } \overline{x}$$
- There is a column $x$ of all 1's: $F = xF_x$
  $$\overline{F} = \overline{x} + \overline{F}_x$$
- There is a column $x$ of all 0's: $F = \overline{x}F_x$
  $$\overline{F} = x + \overline{F}_x$$

If cover is positive (negative) unate at $x$, use

$$F = F_x + x\overline{F}_x \quad (\text{or } F = x\overline{F}_x + F_x)$$

Complementation example

Recursive splitting

$F = \overline{a}b + ac + a' \overline{b}'$

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

$F_a' = 1 \begin{bmatrix} 2 & 2 & 2 \end{bmatrix}$

$F_a' = \emptyset$

$F_{ab'} = c \begin{bmatrix} 2 & 2 & 1 \end{bmatrix}$

$F_{ab'} = c'$

$F_{ab} = 1 \begin{bmatrix} 2 & 2 & 2 \end{bmatrix}$

$F_{ab} = \emptyset$
**Complementation example**

Reconstruct by merging

\[
\begin{align*}
F_{a} &= \begin{bmatrix} 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \\
F_{a}' &= \begin{bmatrix} 2 & 2 & 2 \end{bmatrix} \\
F_{b} &= \begin{bmatrix} 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \\
F_{b}' &= \begin{bmatrix} 2 & 2 & 2 \end{bmatrix} \\
F_{ab} &= c \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \\
F_{ab}' &= \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \\
F_{ab} &= \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \\
F_{ab}' &= \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \\
F' &= ab'c' \\
F'_a &= F_{ab} + b' F_{ab}' = b'c' \\
F'_a &= F_{ab} = 1 \\
F_{ab}' &= c \\
F_{ab} &= 0
\end{align*}
\]

**Expand**

Fundamental procedure of Espresso.

Let \( F \) be a cover of \( f \).

Examine if a cube \( c \) can be expanded (lifted) to, and replaced with, a prime implicant \( d \), such that \( c \subseteq d \).

Each cube \( c \in F \) is replaced by a single prime implicant, so that new cover \( F' \)

\[ | F' | \leq | F | \]
Expand direction

Two problems:

- choice of the cube to be expanded
- the direction of expansion

Blocking matrix guides the expansion of a cube into prime.

\[ f = bcd + abc + abd \]
\[ \bar{f} = \bar{c}d + \bar{a}d + \bar{a}b + \bar{b}c \]

Blocking matrix for \( a\bar{b}c \):

\[
B = \begin{bmatrix}
0 & 0 & 1 & 0 & \bar{c}d \\
1 & 0 & 0 & 0 & \bar{a}d \\
1 & 0 & 0 & 0 & \bar{a}b \\
0 & 0 & 1 & 0 & \bar{b}c
\end{bmatrix} \rightarrow \begin{bmatrix}
0 & 0 & 1 & 0 & \bar{c}d \\
1 & 0 & 0 & 0 & \bar{a}d \\
1 & 0 & 0 & 0 & \bar{a}b \\
0 & 0 & 1 & 0 & \bar{b}c
\end{bmatrix}
\]

How to Expand

Validity of expansion, containment check: is the expanded cube contained in \( F \)?

- Let \( c \) be the cube being expanded, and \( \bar{c} \) be the cube that contains the added minterms. Need to check if: \( \bar{c} \subseteq (F - c) \).

This can be solved by the tautology check:

\[(F - c)\bar{c} \equiv 1\]

Note: this also applies to Reduce operation.
**Expand example**

Example:

\[ f = b\overline{c}d + \overline{a}bc + abd \]

Expand \( abc \rightarrow ac \): test if \( abc \subseteq f \). To do that, check if \( f_{abc} = 1 \)

\[ f_{abc} = \overline{d} + d \neq 1 \]

Is cube \( ac \) is a prime implicant?

Try \( ac \rightarrow a \): test if \( ac \subseteq f \)

\[ f_{ac} = b\overline{d} \neq 1 \]

Try \( ac \rightarrow c \): test if \( ac \subseteq f \)

\[ f_{ac} = b\overline{d} \neq 1 \]

Cannot expand any further, cube \( ac \) is a prime implicant.

---

**Reduce**

Transform a cover of prime implicants

- replace each prime implicant \( p \), where possible, with a smaller, non-prime implicant contained by \( p \).

Purpose of Reduce: iterative improvement

- moves function away from local minimum
- hopefully the subsequent Expand will determine a better set of primes

Similar to Expand (inverse):

![Diagram showing reduction and expansion process]
Redundancy removal

Irredundant (minimal) cover $F$: no proper subset of $F$ is also a cover of $f$.
Extract minimal subset of cubes to cover $f$.
Example:

$$F = ab + ac + bc$$
$$\Rightarrow ab + bc$$

Detecting essential primes

Essential prime: prime implicant that covers a minterm not covered by any other prime implicant. Must be included in any cover of $f$.

Essential primes can be removed from $F$ (put aside) to simplify logic minimization.

Theorem: Let $F$ be written as $G \cup p$, where $p$ is a prime implicant of function $f$, and $G \cap p = \emptyset$. Then, $p$ is an essential prime implicant of $f$ if and only if $p$ is not covered by consensus($G, p$).
**Essential primes - examples**

**Example 1:**

\[ p = ab\overline{cd}, \ G = bd, \ \text{cons}(G,p) = ab\overline{c} \]

\( p \subseteq \text{cons}(G,p) \), hence **not essential**

\[
\begin{array}{cccc}
\text{cd} \\
ab & 00 & 01 & 11 & 10 \\
00 & & & & \\
01 & & 1 & 1 & \\
11 & 1 & 1 & 1 & \\
10 & & & & \\
\end{array}
\]

**Example 2:**

\[ p = ab\overline{c}, \ G = bd, \ \text{cons}(G,p) = abd \]

\( p \not\subseteq \text{cons}(G,p) \), hence **essential**

\[
\begin{array}{cccc}
\text{cd} \\
ab & 00 & 01 & 11 & 10 \\
00 & & & & \\
01 & & & 1 & \\
11 & 1 & 1 & 1 & \\
10 & & & & \\
\end{array}
\]

**Last_gasp and Make_sparse**

**Last_gasp**

- Computes the maximal reduction of every cube of \( ON(f) \).
- Modified Reduce and Expand operations
- Guarantees a **weak form of optimality**:
  * no single prime implicant can be added to the cover such that two primes can be eliminated

**Make_sparse**, final operation

- Attempts to make PLA as sparse as possible (better folding, fewer transistors, fanin, etc.).
ESPRESSO Options

- Heuristic minimization (*espresso*)
- Exact minimization (*espresso -Dexact*)
  Combines ideas of Q-M method and Espresso
  * all prime cubes generated by the method of Espresso
  * covering problem is solved using branch-and-bound techniques
  * the size of a covering table is reduced by extracting essential prime implicants
- Multiple-valued logic minimizer (*espresso -mv*)
  * symbolic minimization
  * constrained state encoding, etc.