Outline

• Review of Boolean algebra.

• Representations of logic functions.

• Matrix representations of covers.

• Operations on logic covers.
• Function $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$.

• **Cofactor** of $f$ with respect to variable $x_i$:
  
  $- f_{x_i} \equiv f(x_1, x_2, \ldots, 1, \ldots, x_n)$.

• **Cofactor** of $f$ with respect to variable $x'_i$:
  
  $- f_{x'_i} \equiv f(x_1, x_2, \ldots, 0, \ldots, x_n)$.

• **Boole’s expansion theorem**:
  
  $- f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i \cdot f_{x_i} + x'_i \cdot f_{x'_i}$
Example

- Function: \( f = ab + bc + ac \)

- Cofactors:
  - \( f_a = b + c \)
  - \( f_{a'} = bc \)

- Expansion:
  - \( f = af_a + a'f_{a'} = a(b + c) + a'bc \)
• 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.
Background

- Function $f(x_1, x_2, \ldots, x_i, \ldots, x_n)$.

- Boolean difference of $f$ w.r.t. variable $x_i$:
  $$\frac{\partial f}{\partial x_i} \equiv f_x \oplus f_{x'}.$$  

- Consensus of $f$ w. r. to variable $x_i$:
  $$C_{x_i} \equiv f_x \cdot f_{x'}.$$  

- Smoothing of $f$ w. r. to variable $x_i$:
  $$S_{x_i} \equiv f_x + f_{x'}.$$
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 \( S_a = f_a + f_{a'} = b + c \).
Generalized expansion

• Given:
  
  – A Boolean function $f$.
  
  – Orthonormal set of functions:
    \[ \phi_i, \quad i = 1, 2, \ldots, k. \]

• Then:
  
  \[ f = \sum_i^k \phi_i \cdot f_{\phi_i} \]
  
  – 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:
  
  - $ab \subseteq f_{\phi_1} \subseteq 1$
  
  - $a'bc + ab'c \subseteq f_{\phi_2} \subseteq ab + bc + ac$.

- Cofactors: $f_{\phi_1} = 1$ and $f_{\phi_2} = a'bc + ab'c$.

\[
\begin{align*}
  f &= \phi_1 f_{\phi_1} + \phi_2 f_{\phi_2} \\
  &= ab \cdot 1 + (a' + b')(a'bc + ab'c) \\
  &= ab + bc + ac
\end{align*}
\]
Generalized expansion theorem

- Given:
  - Two functions \( f \) and \( g \).
  - Orthonormal set of functions:
    \[ \phi_i, \quad i = 1, 2, \ldots, k. \]
  - 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 representations of logic covers

- Used in logic minimizers.

- Different formats.

- Usually one row per implicant.

- Symbols: 0, 1, *. (and other)
The positional cube notation

- Encoding scheme:

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</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>

- Operations:
  - Intersection – AND
  - Union – OR
Example

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

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

\[
\begin{pmatrix}
10 & 10 \\
01 & 01
\end{pmatrix}
\]

- Cofactor w.r. to 01 11:
  - First row – void.
  - Second row – 11 01 .

- Cofactor \( f_a = b \)
Multiple-valued-input functions

• Input variables can have many values.

• Representations:
  – Literals: set of valid values.
  – Sum of products of literals.

• Extension of positional cube notation.

• 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
Operations on logic covers

- **Recursive paradigm:**
  - Expand about a mv-variable.
  - Apply operation to cofactors.
  - Merge results.

- **Unate heuristics:**
  - Operations on unate functions are simpler.
  - Select variables so that cofactors become unate functions.
Tautology

- Check if a function is always TRUE.

- Recursive paradigm:
  - Expand about a mv-variable.
  - If all cofactors are TRUE then function is a tautology.

- Unate heuristics:
  - If cofactors are unate functions additional criteria to determine tautology.
  - Faster decision.
Recursive tautology

- **TAUTOLOGY:** the cover has a row of all 1s. (Tautology cube).

- **NO TAUT.**: the cover has a column of 0s. (A variable that never takes a value).

- **TAUTOLOGY:**
  the cover depends on one variable,
  and there is no column of 0s in that field.

- When a cover is the union of two subcovers, that depend on disjoint subsets 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:

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

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

- No column of 0 – Tautology.
- Cofactor w.r.to $b$ is: 11 11 11.

- *Function is a TAUTOLOGY.*
• Theorem:
  
  – A cover $F$ contains an implicant $\alpha$ iff $F_\alpha$ is a tautology.

• Consequence:

  – Containment can be verified by the tautology algorithm.
Example

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

- Check covering of \( bc \): \( C(bc) = 11\ 01\ 01 \).

- Take the cofactor:

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

- Tautology: \( bc \) is contained by \( f \).
Complementation

• Recursive paradigm:

\[ f' = x \cdot f'_x + x' \cdot f'_{x'} \]

• Steps:

  – Select variable.

  – Compute cofactors.

  – Complement cofactors.

• 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 De Morgan’s law.

- All the 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 $f$ be positive unate: $f' = f'_x + x' \cdot f'_x$.
  
  – If $f$ be negative unate: $f' = x \cdot f'_x + f'_x$.

• Consequence:
  
  – Complement computation is simpler.
  
  – One branch to follow in the recursion.

• Heuristic:
  
  – Select variables to make the cofactors 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 $C(b') = 11 10 11$ yields $11 10 10$.
  
  - $11 10 10$ intersected with $C(a) = 01 11 11$ yields $01 10 10$.

- Complement: $F' = 01 10 10$. 
Example (3)

RECURRENSIVE SEARCH

F_a: = TAUT
COMP = \phi

F_{ab}: = TAUT
COMP = \phi

F_{ab'}: = c
COMP = c'

F_a': = TAUT
COMP = \phi
• Matrix oriented representation:
  – Used in two-level logic minimizer.
  – May be wasteful of space (sparsity).
  – Good heuristics tied to this representation.

• Efficient Boolean manipulation exploits recursion.