Function Definition

- Assume you have a logic function with n-input variables and m-output variables

\[
B = \{0, 1\} \quad \text{//input alphabet is 1 or 0}
\]
\[
Y = \{0, 1, 2\} \quad \text{//output alphabet is 1, 0, or 2 (don't care)}
\]

- Logic function is simply a mapping of input combinations to output values

\[
F : B^n \rightarrow Y^m \text{ where } x = [x_1, ..., x_n] \in B^n \text{ is the input}
\]
\[
y = [y_1, ..., y_m] \text{ is the output of } F
\]

Function Mapping

**Example 1:** \( F_1(a, b) = a' \); \( F_2(a, b) = a + b \)

\[
\begin{array}{c|cc}
 a & b & F_1 & F_2 \\
\hline
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

When \( m \geq 2 \) we have a multiple output function

**Example 2:** \( F_0(a, b, c) = b' + ac \)

\[
\begin{array}{c|ccc}
 a & b & c & F_0 \\
\hline
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

When \( m = 1 \) we have a single output function

On, Off, and DC Sets

- For a given function we can define the \textbf{on-set} \( x^{on} \subseteq B^n \) as the set of input values \( x \) such that \( F(x) = 1 \)

\[
F^{on} = \{ \{0, 0, 0\}, \{0, 0, 1\}, \{1, 0, 0\}, \{1, 0, 1\}, \{1, 1, 1\} \}
\]

- For a given function we can define the \textbf{off-set} \( x^{off} \subseteq B^n \) as the set of input values \( x \) such that \( F(x) = 0 \)

\[
F^{off} = \{ \{0, 1, 0\}, \{0, 1, 1\}, \{1, 1, 0\} \}
\]

- For a given function we can define the \textbf{don't care-set} \( x^{dc} \subseteq B^n \) as the set of input values \( x \) such that \( F(x) = 2 \)

\[
F^{dc} = \{ \} \quad \text{//empty}
\]
On, Off, and DC Sets

**Example 3:** Given the following truth table, identify $F_{ON}$, $F_{OFF}$, and $F_{DC}$

\[
\begin{array}{ccc}
 F_{ON} &=& \{ (0, 0, 0), (0, 0, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0) \} \\
 F_{OFF} &=& \{ (0, 1, 0), (0, 1, 1) \} \\
 F_{DC} &=& \{ (1, 1, 1) \}
\end{array}
\]

Completely Specified Functions

- A completely specified function (c.s.f) is a function where all values of the input map to a 1 or 0 (i.e. no don’t care conditions)

\[
\begin{array}{cccccccc}
 a & b & c & F_{ON} & F_{OFF} & F_{DC} \\
 0 & 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 & 1 & 1 \\
 0 & 1 & 0 & 0 & 1 & 0 \\
 0 & 1 & 1 & 1 & 0 & 1 \\
 1 & 0 & 0 & 0 & 1 & 0 \\
 1 & 0 & 1 & 1 & 0 & 1 \\
 1 & 1 & 0 & 0 & 1 & 1 \\
 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

Ex 1 and 2 are completely specified functions

Ex 3 is not a completely specified function

Logic Functions Operations & Definitions

**Complement** of a c.s.f. (completely specified function) $F'$ is defined as $F'_{ON} = F_{OFF}$ and $F'_{OFF} = F_{ON}$ (i.e. switch on and off sets)

\[
\begin{array}{cccccc}
 a & b & c & F & F' \\
 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 & 1 \\
 0 & 1 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 1 \\
 1 & 0 & 0 & 1 & 0 \\
 1 & 0 & 1 & 1 & 0 \\
 1 & 1 & 0 & 0 & 1 \\
 1 & 1 & 1 & 0 & 1
\end{array}
\]

**Intersect** (or product) of two c.s.f. $F$ and $G$, denoted as $F \land G$, is the c.s.f. $H$ where $H_{ON} = F_{ON} \land G_{ON}$ (i.e. must be in both)

\[
\begin{array}{cccccc}
 a & b & c & F & G & H \\
 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 1 & 1 \\
 1 & 0 & 0 & 0 & 1 & 0 \\
 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

**Difference** between two c.s.f. $F$ and $G$, denoted as $F - G$, is the c.s.f. $H$ where $H_{ON} = F_{ON} \setminus G_{ON}$ (i.e. it's in $F$ but not in $G$)

\[
\begin{array}{cccccc}
 a & b & c & F & G & H \\
 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 1 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 \\
 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\]
Logic Functions Operations & Definitions

Union (or sum) of two c.s.f. F and G, denoted F + G or \( F \cup G \), is the c.s.f. where \( H_{ON} = F_{ON} \cup G_{ON} \) (i.e. it's in either F or G).

A tautology is a c.s.f. whose off-set is empty, written \( F = 1 \) (i.e. function always evaluates to 1).

\[ F \cup D \cup R \] is a tautology if \( F, D, R \) are mutually disjoint (no elements in common).

Cubes and Covers

- Function can also be represented as a cube in a Boolean n-space.
- Each vertex represents a value of the input and used to specify which components of F it belongs to (ON, OFF, or DC).

Example 4: \( F_1(a, b) = \sum \{1, 2, 3\} \)

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

Example 5: \( F_1(x_1, x_2, x_3) = \sum \{0, 1, 4, 5, 6\} \) + \( \varepsilon \{7\} \)

\( F_2(x_1, x_2, x_3) = \sum \{0, 2, 3, 6, 7\} \) + \( \varepsilon \{5\} \)

Cubes and Covers

- We want to provide this information in a more compact form.

Let \( p \) be a product term associated with a S.O.P expression with \( n \) inputs and \( m \) outputs.

Then a cube \( p \) is specified by a row vector \( c = [c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}] \) where:

- \( c_i = 0 \) if \( x_i \) appears complemented in \( p \) for \( i=1, \ldots, n \)
- \( 1 \) if \( x_i \) appear NOT complemented in \( p \) for \( i=1, \ldots, n \)
- \( 2 \) if \( x_i \) does not appear in \( p \) for \( i=1, \ldots, n \)
- \( 3 \) if \( p \) is NOT present in the algebraic representation of \( F \)
- \( 4 \) if \( p \) is present in the algebraic representation of \( F \)
Compact Cubical Form

Example 6:

\[ F_1 = B' + AC' \]
\[ F_2 = B + AC \]

How do we represent \( B' \) in compact cubical form?

\[ c = [ 2 \ 0 \ 4 \ 3 ] \]

- Is \( B' \) a product term in \( F_2? \) No, \( c_i = 3 \)
- Is \( B' \) a product term in \( F_1? \) Yes, \( c_i = 4 \)
- How does \( C \) appear in \( F_1? \) \( C \) does not appear, \( c_i = 2 \)
- How does \( B \) appear in \( F_1? \) \( B \) is complemented, \( c_i = 0 \)
- How does \( A \) appear in \( F_1? \) \( A \) does not appear, \( c_i = 3 \)

Compact cubical form represents vertices of a cube corresponding to a product term, to represent the remainder of \( F_1 \) and \( F_2 \)

\[ F = \begin{bmatrix} 2 & 0 & 2 & 4 & 3 \end{bmatrix} \quad // B' \]
\[ \begin{bmatrix} 1 & 2 & 0 & 4 & 3 \end{bmatrix} \quad // AC' \]
\[ \begin{bmatrix} 2 & 1 & 3 & 4 \end{bmatrix} \quad // B \]
\[ \begin{bmatrix} 1 & 2 & 1 & 3 & 4 \end{bmatrix} \quad // AC \]

This matrix representation is the input used by Espresso

This set of cubes represents a cover

Boolean n-space vs. Compact Cubical Form

\( B' \) as vertices in a Boolean n-space =
\( \{ [0, 0, 0], [0, 0, 1], [1, 0, 0], [1, 0, 1] \} \)

Graphical representation of \( B' \)

\( B' \) in compact cubical form = [2 0 2 4 3]

All vertices with a 0 in second coordinate represented by this one cube

Operations on Cubes

**Intersect** (or product) of two cubes, written as \( c \cap d \) or \( cd \), is the cube \( e \) given by the following table

<table>
<thead>
<tr>
<th>( c_i )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( \Phi )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Basic idea: want an input cube whose input part corresponds to vertices common to \( c \) and \( d \), resulting cube output part also represents when \( c \) and \( d \) are both present

\( c_i \cap d_i = e_i = [2 \ \Phi \ 2 \ 3 \ 4] \)

When there is an index = \( \Phi \), or if the output part is all 3’s, the cube is empty and we can get rid of it
Operations on Cubes

Intersection of two sets of cubes is the set obtained by performing pair wise intersection of all cubes in the two sets

Example 6:

\[
\begin{align*}
\{20243\} \cap \{11144\} &= \{1\Phi 143\} \\
\{20243\} \cap \{00043\} &= \{00043\} \\
\{20243\} \cap \{20234\} &= \{20234\} \\
\{11043\} \cap \{11144\} &= \{11\Phi 43\} \\
\{11043\} \cap \{00043\} &= \{\Phi \Phi 043\} \\
\{11043\} \cap \{20234\} &= \{1\Phi 033\} \\
\{02234\} \cap \{11144\} &= \{\Phi 1134\} \\
\{02234\} \cap \{00043\} &= \{00033\} \\
\{02234\} \cap \{20234\} &= \{00234\}
\end{align*}
\]

Operations on Cubes

Union (or sum) of two cubes, written as \(c+d\) or \(c+d\), is the set of vertices covered by the input part of either \(c\) or \(d\)

Basic idea: combine all cubes

\[
c_c = \begin{bmatrix} 2 & 0 & 2 & 4 & 3 \end{bmatrix} \\
d_d = \begin{bmatrix} 1 & 2 & 0 & 4 & 3 \end{bmatrix}
\]

\[
c_c \cup d_d = \begin{bmatrix} 2 & 0 & 2 & 4 & 3 \\ 1 & 2 & 0 & 4 & 3 \end{bmatrix}
\]

Operations on Cubes

Intersection of two sets of cubes is the set obtained by performing pair wise intersection of all cubes in the two sets

Example 6:

\[
\begin{align*}
\{20243\} \cap \{11144\} &= \{1\Phi 143\} \\
\{20243\} \cap \{00043\} &= \{00043\} \\
\{20243\} \cap \{20234\} &= \{20234\} \\
\{11043\} \cap \{11144\} &= \{11\Phi 43\} \\
\{11043\} \cap \{00043\} &= \{\Phi \Phi 043\} \\
\{11043\} \cap \{20234\} &= \{1\Phi 033\} \\
\{02234\} \cap \{11144\} &= \{\Phi 1134\} \\
\{02234\} \cap \{00043\} &= \{00033\} \\
\{02234\} \cap \{20234\} &= \{00234\}
\end{align*}
\]

Operations on Cubes

What about Complement?

Use DeMorgan's Law to determine what happens to a product term when complemented

\[(ab)' = a' + b'\]

When there is an \(i\) in \(c\), or if the output part is all 3's, the cube is empty and we can get rid of it

\[
\text{Solution} = \begin{bmatrix} 0 & 0 & 0 & 4 & 3 \\ 0 & 0 & 2 & 3 & 4 \end{bmatrix}
\]

ECE 474a/575a 17 of 25

ECE 474a/575a 18 of 25

ECE 474a/575a 19 of 25

ECE 474a/575a 20 of 25
Operations on Cubes

What about Complement?

Use DeMorgan’s Law to determine what happens to a product term when complemented:

\[(ab)' = a' + b'\]

\[F(a, b, c) = a'bc\]

\[F'(a, b, c) = (a'bc)'

\[= a + b' + c'\]

\[
\begin{bmatrix}
0 & 1 & 2 & 2 & 2 & 1 & 4 & 4 & 3 & 4 \\
2 & 0 & 0 & 4 & 3 & 4 & 1 & 0 & 1 & 3 \\
2 & 2 & 0 & 0 & 4 & 3 & 4 & 1 & 0 & 1 \\
\end{bmatrix}
\]

Many other transformations exist (distance, consensus, etc.) we’ll stick with the basic ones for now.

Espresso Optimization Goal

- Espresso algorithm returns a “minimized cover”
- What is the algorithm trying to minimize?

\[\mathcal{F} = (NPT, NLI, NLO)\]

- \(NLO\) - # of literals in output part
- \(NLI\) - # of literals (non-2’s) in input part of cover
- \(NPT\) - # of product terms in a cover

Espresso Subroutine

- Many smaller subroutines used in Espresso
- We will only cover a few
  - Unwrap
  - Unate Complement
  - Complement
  - Expand

Unwrap(F)

- Incoming data may have output sharing
- Apply Unwrap(F) to the input so we start with a less biased starting point
- When complete algorithm can decide what sharing is desirable

\[
\begin{bmatrix}
1 & 2 & 4 & 4 \\
2 & 0 & 4 & 3 \\
2 & 0 & 1 & 3 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 & 4 & 3 \\
1 & 2 & 3 & 4 \\
2 & 0 & 4 & 3 \\
2 & 0 & 1 & 3 \\
\end{bmatrix}
\]

Each cube feeding \(k\) different outputs are replaced with \(k\) cube feeding 1 output.
Coming Soon …

- Unate Complement
- Complement
- Expand