ECE 595Z
Digital Systems Design Automation

Module 2 (Lectures 3-5) : Advanced Boolean Algebra
Lecture 5

Anand Raghunathan
MSEE 348
raghunathan@purdue.edu
Terminology Checklist

• Boolean Algebra
• Boolean Function
• Cube
• Implicant (of a function)
• Minterm
• Cover (of a function)
• Tautology
• Satisfiable / Un-satisfiable
• Sum-of-products
• Minterm canonical representation
• Product-of-sums
• Conjunctive Normal Form
• Disjunctive Normal Form
• Binary Decision Tree
• Binary Decision Diagram
• Symbolic Simulation
Lecture #5 Outline

• Converting between representations - continued
• Co-factors and their applications
Conversion: Example #2

• How do you convert a general Boolean network (multi-level circuit) into a Boolean formula that is linear in the circuit size?
  – Size(formula) = \( O(M) \) where \( M \) = no. of gates in circuit
  – SOP may be exponential in the worst case
  – Hints
    • Use variables to represent intermediate signals in the circuit
    • Compose the formula using a 1 : 1 mapping from each gate in the circuit into a piece of the formula
Converting a Boolean Circuit into a CNF Formula

• First, let us see how very simple circuits (single gates) can be expressed in CNF form:

\[ c = ab \]
\[ c \rightarrow ab, \ ab \rightarrow c \]
\[ a = 0 \rightarrow c = 0 \]
\[ b = 0 \rightarrow c = 0 \]
\[ a = 1, \ b = 1 \rightarrow c = 1 \]

\[ (a + c')(b + c')(a' + b' + c) \]
Converting a Boolean Circuit into a CNF Formula

- Simple rules for converting various basic gates into CNF equivalent

<table>
<thead>
<tr>
<th>Gate Type</th>
<th>Function</th>
<th>CNF Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>$c = a'$</td>
<td>$(a+c)(a'+c')$</td>
</tr>
<tr>
<td>AND</td>
<td>$c=ab$</td>
<td>$(a+c')(b+c')(a'+b'+c)$</td>
</tr>
<tr>
<td>NAND</td>
<td>$c=a'+b'$</td>
<td>$(a+c)(b+c)(a'+b'+c')$</td>
</tr>
<tr>
<td>OR</td>
<td>$c=a+b$</td>
<td>$(a'+c)(b'+c)(a+b+c')$</td>
</tr>
<tr>
<td>NOR</td>
<td>$c = a'b'$</td>
<td>$(a'+c')(b'+c')(a+b+c)$</td>
</tr>
</tbody>
</table>
Converting a Boolean Circuit into a CNF Formula

- Now, we are ready to convert a multi-level circuit into a CNF formula
  - Simply concatenate formulae representing each of its gates

\[(a+j)(b+j)(a'+b'+j')\]
\[(c+k)(c'+k')\]
\[(d+l)(d'+l')\]
\[(e+m)(f+m)(e'+f'+m')\]
\[(m+o)(m'+o')\]
\[(j+n)(k+n)(j'+k'+n')\]
\[(n+p)(l+p)(n'+l'+p')\]
\[(o+q)(g+q)(o'+g'+q')\]
\[(q+r)(q'+r')\]
\[(p+s)(r+s)(p'+r'+s')\]
\[(s+u)(s'+u')\]
\[(u+v)(h+v)(u'+h'+v')\]
\[(q+t)(q'+t')\]
\[(t+w)(i+w)(t'+l'+w')\]

Known as the Tseitin Transformation
Co-factors

• A very useful operation on Boolean functions

• Applications of co-factors
  – Shannon’s expansion
  – Boolean difference
  – Universal and Existential Quantification
Co-factors of Boolean Functions

A co-factor of a function is derived by fixing one of the variables to a constant (0 or 1), resulting in a new function of n-1 variables.

Given a function \( f(x_1 \ldots x_n) \)
- Positive co-factor w.r.t. \( x_i \) is defined as
  \[
  f_{x_i}(x_1 \ldots x_{i-1}, x_{i+1} \ldots x_n) = f(x_1 \ldots x_{i-1}, x_i = 1, x_{i+1} \ldots x_n)
  \]
- Negative co-factor w.r.t. \( x_i \) is defined as
  \[
  f_{x_i'}(x_1 \ldots x_{i-1}, x_{i+1} \ldots x_n) = f(x_1 \ldots x_{i-1}, x_i = 0, x_{i+1} \ldots x_n)
  \]

Examples:
\[
\begin{align*}
f &= ab + bc + ac \\
f_a &= 1.b + bc + 1.c = b + c \\
f_{a'} &= \\
f_b &= \\
f_{b'} &= \\
f_c &= \\
f_{c'} &= 
\end{align*}
\]
Co-factors of Boolean Functions

• Also called
  – Shannon co-factors
  – Restriction of a function on a variable

• Can be applied on multiple variables
  \[ f_{x_i x_j} = f(x_1 \ldots x_i = 1 \ldots x_j = 0 \ldots x_n) \]

• Order does not matter
  \[ f_{x_i x_j} = (f_{x_i})_{x_j} = (f_{x_j})_{x_i} \]

• Co-factor w.r.t. a cube

Examples:

\[ f = ab + bc + ac \]

\[ f_{ab} = \]
\[ f_{ab'} = \]
\[ f_{a'b'c'} = \]
\[ f_{ab'c} = \]

\[ g_{ab} = \]
\[ g_{a'b} = \]
\[ g_{b'c'} = \]
\[ g_{abc'} = \]
OK, so why do we need Co-factors?

- Many applications... for example
- Recall **Taylor series** from high-school math?
  - A representation of a (real or complex) function as a sum of polynomial terms (1, x, x^2, x^3, x^4, ...)
    - \( e^x = 1 + x + x^2/2! + x^3/3! + ... \)
  - General form:
    \[ f = \]

- Question: Is there a similar concept for Boolean functions?
Shannon’s Expansion Theorem

- Given a Boolean function $f(x_1 \ldots x_n)$ and any variable $x_i$

$$f = x_i f_{x_i} + x_i' f_{x_i'}$$

Structural view of Shannon Expansion

Diagram showing the expansion of $f$ with inputs $x_1, x_2, \ldots, x_i, \ldots, x_n$ and their corresponding expansions with and without $x_i$. The output is connected to an OR gate with inputs $0$ and $1$.
Shannon Expansion

- Also called **Shannon Decomposition**
- Can be applied recursively to "decompose" a function into its co-factors
  - In the extreme case, just a network of multiplexers

Shannon Expansion

• Example

\[ f = xy + zw' + x'w' \]
Properties of Co-factors

• Given two functions $f(x)$ and $g(x)$

• How can we compute co-factors of a function $h$ that is derived from $f$ and $g$?

<table>
<thead>
<tr>
<th>Function</th>
<th>Co-factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x) = f'(x)$</td>
<td>$h_{x_i} = (f_{x_i})'$</td>
</tr>
<tr>
<td></td>
<td>$h_{x_i} = (f_{x_i})'$</td>
</tr>
<tr>
<td>$h(x) = f(x) \text{ AND } g(x)$</td>
<td>$h_{x_i} = f_{x_i} \text{ AND } g_{x_i}$</td>
</tr>
<tr>
<td></td>
<td>$h_{x_i} = f_{x_i} \text{ AND } g_{x_i}$</td>
</tr>
<tr>
<td>$h(x) = f(x) \text{ OR } g(x)$</td>
<td>$h_{x_i} = f_{x_i} \text{ OR } g_{x_i}$</td>
</tr>
<tr>
<td></td>
<td>$h_{x_i} = f_{x_i} \text{ OR } g_{x_i}$</td>
</tr>
<tr>
<td>$h(x) = f(x) \text{ XOR } g(x)$</td>
<td>$h_{x_i} = f_{x_i} \text{ XOR } g_{x_i}$</td>
</tr>
<tr>
<td></td>
<td>$h_{x_i} = f_{x_i} \text{ XOR } g_{x_i}$</td>
</tr>
</tbody>
</table>

Co-factor of complement is complement of co-factor
Co-factor of AND is AND of co-factors
Co-factor of OR is OR of co-factors
Co-factor of XOR is XOR of co-factors

The co-factor operation distributes over any binary operator
Combinations of Co-factors

- Combining $f_x$ and $f_x'$ in different ways leads to useful new functions
  - $f_x \oplus f_x' = ?$
  - $f_x \cdot f_x' = ?$
  - $f_x + f_x' = ?$
Another analogy to the “real” world

- The derivative of a function measures how much it changes when it’s input changes.

- Let us think of the analogy in the case of Boolean functions (which only take values 0 and 1).

\[ f'(x) = \lim_{\Delta \to 0} \frac{f(x + \Delta) - f(x)}{\Delta} \]
Boolean Difference

- Boolean difference of a function w.r.t. a variable is the exclusive-OR of the Shannon co-factors w.r.t. the variable.

\[ \frac{\partial f}{\partial x} = f_x \oplus f_{x'} \]

- Interpretation: \( \frac{\partial f}{\partial x} = 1 \) → \( f \) is sensitive to the value of \( x \)

- A new function that does not depend on \( x \)

Example:
\[ f = xy + zw' + x'w' \]
\[ f_x = \]
\[ f_{x'} = \]
\[ \frac{\partial f}{\partial x} = \]
Boolean Difference

• Examples:

\[
\frac{\partial s}{\partial a} = \frac{\partial c_{out}}{\partial c_{in}} = \\
\frac{\partial out}{\partial a} = \frac{\partial out}{\partial s} =
\]

Full Adder

\[
\begin{align*}
a & \rightarrow s \\
b & \rightarrow s \\
c_{in} & \rightarrow \text{cout}
\end{align*}
\]
Application of Boolean Difference

- **Manufacturing test**
  - Apply test vectors to ensure that each fabricated instance of an IC behaves correctly
  - Cannot apply exhaustive test set (too big!)

- **Fault model:** Abstraction of physical defects that could impact the IC
  - Commonly used: “stuck-at” fault model
  - Signals in the circuit are stuck-at-0, stuck-at-1

How do you derive a test vector to detect the fault c s-a-0?

(i) Set c = 1
(ii) Set other inputs such that output of good and faulty circuits are different

Looks familiar?
Co-factors: Re-cap

• A very useful operation on Boolean functions
  – Derived by fixing one of the variables to a constant (0 or 1)

• Applications of co-factors
  – Shannon’s expansion – a way to recursively simplify or divide Boolean functions
  – Boolean difference \((f_x \oplus f_x')\)
  – Universal and Existential Quantification
Quantification

- Two more functions of Shannon cofactors
  - $f_{x_i} \cdot f_{x_i'} = 1$ specifies when $f = 1$ independent of the value of $x_i$
    
    $f(x_1 \ldots x_{i-1}, x_i = 1, x_{i+1} \ldots x_n) = 1$ AND
    $f(x_1 \ldots x_{i-1}, x_i = 0, x_{i+1} \ldots x_n) = 1$

- Called Universal quantification or Consensus

\[
\forall x(f) = f_x \cdot f_{x'}
\]

\[
C_a(f)
\]