Multi-level Logic Optimization: Outline

Overview of Multi-level Optimization: An Example

Core Concepts:
- Boolean Function Decomposition
- Boolean and Algebraic Division
- Identifying Divisors
Representation: Boolean Network

Nodes labeled with SOP expressions

Boolean Network, Explained

It’s a graph:
- Primary inputs (variables)
- Primary outputs
- Intermediate nodes (in SOP form in terms of its inputs)

- Quality of network: area, delay, …
  - measured in terms of #(litersals), depth, …
Tech.-Independent Multi-Level Optimization: Operations on Boolean Network

Involves performing the following operations "iteratively" until "good enough" result is obtained:

1. **Simplification**
   Minimizing two-level logic function (SOP for a single node)

2. **Elimination**
   Substituting one expression into another.

3. **Decomposition**
   Expressing a single SOP with 2 or more simpler forms

4. **Extraction**
   Finding & pulling out subexpressions common to many nodes

5. **Substitution**
   Like extraction, but nodes in the network are re-used

---

Example (due to G. De Micheli)

\[
\begin{align*}
    v &= a'd + bd + c'd + ae' \\
    p &= ce + de \\
    r &= p + a' \\
    s &= r + b' \\
    t &= ac + ad + bc + bd + e \\
    q &= a + b \\
    u &= q'c + qc' + qc \\
    w &= \text{expression} \\
    x &= \text{expression} \\
    y &= \text{expression} \\
    z &= \text{expression}
\end{align*}
\]

#literals = 33, depth = 3
Example: Elimination

\[
v = a'd + bd + c'd + ae'
\]
\[
p = ce + de
\]
\[
r = p + a'
\]
\[
s = r + b'
\]
\[
t = ac + ad + bc + bd + e
\]
\[
q = a + b
\]
\[
q = a + b
\]
\[
u = q'c + qc' + qc
\]

#literals = 33, depth = 3

Example: Eliminate node \( r \)

\[
v = a'd + bd + c'd + ae'
\]
\[
p = ce + de
\]
\[
s = p + a' + b'
\]
\[
t = ac + ad + bc + bd + e
\]
\[
q = a + b
\]
\[
u = q'c + qc' + qc
\]

#literals = 32, depth = 2
Example: Simplification

\[ v = a'd + bd + c'd + ae' \]
\[ p = ce + de \]
\[ s = p + a' + b' \]
\[ t = ac + ad + bc + bd + e \]
\[ q = a + b \]
\[ u = qc' + qc + q'c \]

#literals = 32, depth = 2

Example: Simplifying node u

\[ v = a'd + bd + c'd + ae' \]
\[ p = ce + de \]
\[ s = p + a' + b' \]
\[ t = ac + ad + bc + bd + e \]
\[ q = a + b \]
\[ u = q + c \]

#literals = 28, depth = 2
Example: Decomposition

\[
v = a'd + bd + c'd + ae'
\]
\[
p = ce + de
\]
\[
s = p + a' + b'
\]
\[
t = ac + ad + bc + bd + e
\]
\[
q = a + b
\]
\[
u = q + c
\]

#literals = 28, depth = 2

Example: Decomposing node v

\[
v = jd + ae'
\]
\[
p = ce + de
\]
\[
s = p + a' + b'
\]
\[
t = ac + ad + bc + bd + e
\]
\[
q = a + b
\]
\[
u = q + c
\]

#literals = 27, depth = 2
Example: Extraction

\[
\begin{align*}
  j &= a' + b + c' \\
v &= jd + ae' \\
p &= ce + de \\
s &= p + a' + b' \\
t &= ac + ad + bc + bd + e \\
qu &= a + b \\
u &= q + c \\
\end{align*}
\]

\#literals = 27, depth = 2

Example: Extracting from \( p \) and \( t \)

\[
\begin{align*}
  j &= a' + b + c' \\
v &= jd + ae' \\
p &= ke \\
s &= p + a' + b' \\
k &= c + d \\
t &= ka + kb + e \\
qu &= a + b \\
u &= q + c \\
\end{align*}
\]

\#literals = 23, depth = 3
Example: What next? Can we improve further?

\[
v = jd + ae' \\
p = ke \quad s = p + a' + b' \\
k = c + d \quad t = ka + kb + e \\
q = a + b \quad u = q + c \\
\]

#literals = 23, depth = 3

Which Operations Do We Know How to Do?

1. **Simplification**
   Minimizing two-level logic function (SOP for a single node)

2. **Elimination**
   Substituting one expression into another.

3. **Decomposition**
   Expressing a single SOP with 2 or more simpler forms

4. **Extraction**
   Finding & pulling out subexpressions common to many nodes

5. **Substitution**
   Like extraction, but nodes in the network are re-used
Decomposition by Factoring/Division

Starting with a SOP Form
\[ f = ac + ad + bc + bd + ae' \]
We want to generate an equivalent Factored form
\[ f = (a + b) (c + d) + ae' \]

Reason: Factored forms are ‘natural’ multi-level representations – tree-like expressions

To do factoring, we need to
- Identify divisors
- Perform division

Divisors and Decomposition

Given Boolean function \( F \), we want to write it as
\[ F = D \cdot Q + R \]
where \( D \) – Divisor, \( Q \) – Quotient, \( R \) – Remainder

Decomposition: Searching for divisors which are common to many functions in the network
- identify divisors which are common to several functions
- introduce common divisor as a new node
- re-express existing nodes using the new divisor
Topics

What is division?
- Boolean vs. Algebraic

How to perform division

How to identify divisors

Boolean Division

Given Boolean function $F$, we want to write it as
$$F = D \cdot Q + R$$

**Definition:**

$D$ is a **Boolean divisor** of $F$ if $Q$ and $R$ exist such that
$$F = DQ + R, \quad DQ \neq 0. \quad (F \neq 0)$$

$D$ is said to be a **factor** of $F$ if, $D$ is a divisor of $F$ and in addition, $R = 0$; i.e., $F = DQ$. 
Boolean Division: Key Results

- D is a factor of F iff F.D' = 0
  - ON-SET(D) contains ON-SET(F)

- F ∙ D != 0 iff D is a divisor of F

- How many possible factors D can there be for a given F?

Boolean Division: Proof Ideas

D is a factor of F iff F ∙ D' = 0

- (only if part): F = DQ, so F.D' = 0
- (if part): Given that F.D' = 0, F ⊆ D, so F = DF, or F = D(F+X) where X.D = 0.
  Thus, F = DH for some H.

F ∙ D != 0 iff D is a divisor of F

- (if): F = DQ + R, FD = DQ + DR, since DQ != 0, FD != 0
- (only if): FD != 0 and F = FD + FD', take Q=F+d, R=FD', where dD = 0.

How many possible factors D can there be for a given F?

- Doubly exponential in number of variables
Algebraic Model

Idea: Perform division using only the rules (axioms) of real numbers, not all of Boolean algebra

Real Numbers
\[ a \cdot b = b \cdot a \]
\[ a + b = b + a \]
\[ a \cdot (b \cdot c) = (a \cdot b) \cdot c \]
\[ a + (b + c) = (a + b) + c \]
\[ a \cdot (b + c) = a \cdot b + a \cdot c \]
\[ a \cdot 1 = a \quad a \cdot 0 = 0 \quad a + 0 = a \]

Boolean Algebra
\[ a \cdot b = b \cdot a \]
\[ a + b = b + a \]
\[ a \cdot (b \cdot c) = (a \cdot b) \cdot c \]
\[ a + (b + c) = (a + b) + c \]
\[ a \cdot (b + c) = a \cdot b + a \cdot c \]
\[ a \cdot 1 = a \quad a \cdot 0 = 0 \quad a + 0 = a \]
\[ a + (b \cdot c) = (a + b) \cdot (a + c) \]
\[ a + a' = 1 \quad a \cdot a' = 0 \quad a \cdot a = a \]
\[ a + a = a \]
\[ a + 1 = 1 \quad a + ab = a \quad a \cdot (a + b) = a \]
\[
\begin{align*}
\text{f} &= ab + a'x + b'y \\
\text{as} &\quad \text{f} = ab + dx + ey
\end{align*}

Algebraic Division

- A literal and its complement are treated as unrelated
  - Each literal as a fresh variable
  - E.g.
    \[ f = ab + a'x + b'y \quad \text{as} \quad f = ab + dx + ey \]

- Treat SOP expression as a polynomial
  - Division/factoring then becomes polynomial division/factoring

- Boolean identities are ignored
  - Except in pre-processing
  - Simple local simplifications like \( a + ab \rightarrow a \) performed
Algebraic vs. Boolean factorization

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

Algebraic factorization produces
\[ f = a(\overline{b} + \overline{c}) + \overline{a}(b + c) + b\overline{c} + c\overline{b} \]

Boolean factorization produces
\[ f = (a + b + c)(\overline{a} + \overline{b} + \overline{c}) \]

Algebraic Division Example

\[ F = ac + ad + bc + bd + ae \]

Want to get \( Q, R \), where \( F = D Q + R \) for
1. \( D = a + b \)
2. \( D = a \)
Algebraic Division Algorithm

What we want:
Given F, D, find Q, R
F, D expressed as sets of cubes (same for Q, R)

Approach:
For each cube C in D {
    let B = {cubes in F contained in C}
    if (B is empty) return Q = { }, R = F
    let B = {cubes in B with variables in C removed}
    if (C is the first cube in D we’re looking at)
        let Q = B;
    else Q = Q \ B;
}
R = F \ (Q x D);

Complexity?

Taking Stock

- What we know:
  - How to perform Algebraic division given a divisor D

- What we don’t
  - How to find a divisor D?

- Recall what we wanted to do:
  Given 2 functions F and G, find a common divisor D and factorize them as
    F = D Q1 + R1
    G = D Q2 + R2
New Terminology: Kernels

A kernel of a Boolean expression $F$ is a **cube-free expression** that results when you divide $F$ by a single cube
- That “single cube” is called a co-kernel

**Cube-free expression**: Cannot factor out a single cube that leaves behind no remainder

Examples: Which are cube-free?
- $F_1 = a + b$
- $F_2 = abc + abd$

Kernels: Examples

$F = ae + be + cde + ab$

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Co-kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a,b,cd}$</td>
<td>$e$</td>
</tr>
<tr>
<td>${e,b}$</td>
<td>$?$</td>
</tr>
<tr>
<td>$?$</td>
<td>$b$</td>
</tr>
<tr>
<td>${ae,be,cde,ab}$</td>
<td>$?$</td>
</tr>
</tbody>
</table>

Note: can view kernels as sets of cubes
Why are Kernels Useful?

Multi-level logic optimizer wants to find common divisors of two (or more) functions $f$ and $g$

**Theorem:** [Brayton & McMullen]

$f$ and $g$ have a non-trivial (multiple-cube) common divisor $d$ if and only if there exist kernels $k_f \in K(f)$, $k_g \in K(g)$ such that $k_f \cap k_g$ is non-trivial, i.e., not a cube

(here set intersection is applied to the sets of cubes in $k_f$ and $k_g$)

$: :$ can use kernels of $f$ and $g$ to locate common divisors

---

Theorem, Sketched Informally

$$F = D_1 \cdot K_1 + R_1$$
$$G = D_2 \cdot K_2 + R_2$$

$$K_1 = (X + Y + \ldots) + \text{stuff1}$$
$$K_2 = (X + Y + \ldots) + \text{stuff2}$$

Then,

- $F = (X + Y + \ldots) \cdot D_1 + \text{stuff3}$
- $G = (X + Y + \ldots) \cdot D_2 + \text{stuff4}$

So, if we find kernels and intersect them, the intersection gives us our common divisor
### Kernel Intersection: Example

\[ F = ae + be + cde + ab \]
\[ G = ad + ae + bd + be + bc \]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Co-kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b, cd}</td>
<td>e</td>
</tr>
<tr>
<td>{e, b}</td>
<td>a</td>
</tr>
<tr>
<td>{e, a}</td>
<td>b</td>
</tr>
<tr>
<td>{ae, be, cde, ab}</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Co-kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a, b}</td>
<td>d or e</td>
</tr>
<tr>
<td>{d, e}</td>
<td>a or b</td>
</tr>
<tr>
<td>{d, e, c}</td>
<td>b</td>
</tr>
<tr>
<td>{ad, ae, bd, be, bc}</td>
<td>1</td>
</tr>
</tbody>
</table>

### How do we find Kernels?

**Overview:** Given a function \( F \)

1. **Pick a variable** \( x \) appearing in \( F \), and use it as a divisor
2. **Find the corresponding kernel** \( K \) if one exists (at least 2 cubes in \( F \) contain \( x \))
   - If not, go back to (1) and pick another variable
3. **Use** \( K \) **in place of** \( F \) **and recurse to find kernels of** \( K \)
   - \( F = xK + R \) and \( K = yM + S \) \( \Rightarrow \) \( F = xyM + \ldots \)
   - Add kernels of \( K \) to those of \( F \)
4. **Go back to (1) and pick another variable to keep finding kernels**
Finding Kernels: Example

\[ F = abc + abd + bcd \]

**Intersection is bc.**

Recurse on \( F/bc = a + d \)

---

Take intersection of all cubes containing a variable
Kernel Finding Algorithm

FindKernels(F) {
    K = { };  
    for (each variable x in F) {
        if (F has at least 2 cubes containing x) {
            let S = {cubes in F containing x};  
            let c = cube resulting from intersection of all cubes in S  
            K = K \cup FindKernels(F/c);  //recursion  
        }  
    }  
    K = K \cup F ;  
    return K;  
}

Strong (or Boolean) Division  [optional reading]

Given a function \( f \) to be strong divided by \( g \)

- Add an extra input to \( f \) corresponding to \( g \),
  namely \( G \) and obtain function \( h \) as follows
  
  \[
  h_{DC} = G \overline{g} + \overline{G} g \ 
  \text{Inputs to f that cannot occur}
  
  h_{ON} = f_{ON} - h_{DC} = f_{ON} \cdot \overline{h_{DC}}
  
  h_{OFF} = f_{ON} + h_{DC}
  
  \]

Get: \( h = Q G + R \)

Minimize \( h \) using two-level minimizer
Reading