Division

- Definition:

  G is an algebraic factor of F if there exists an algebraic expression H such that F = GH (using algebraic multiplication)

- Definition:

  G is an Boolean factor of F if there exists an expression H such that F = GH (using Boolean multiplication)

- Example
  - f = ac + ad + bc + bd
    - (a+b) is an algebraic factor of f since f = (a+b)(c+d)
  - f = ¬ab + ac + bc
    - (a+b) is a Boolean factor of f since f = (a+b)(¬a+c)

Why Algebraic Methods?

- Algebraic methods provide fast algorithms for various operations
  - Treat logic functions as polynomials
  - Fast algorithms for polynomials exist
  - Lost of optimality but results are still good
  - Can iterate and interleave with Boolean operations
    - In specific instances, slight extensions are available to include Boolean methods

Weak Division

- **Weak division** is a specific example of algebraic division

- Definition: Given two algebraic expressions F and G, a division is called a weak division if
  1. it is algebraic and
  2. R has as few cubes as possible
  - The quotient H resulting from weak division is denoted by F/G

- Theorem: Given expressions F and G, H and R generated by weak division are unique

Weak Division

```
ALGORITHM WEAK_DIV(F,G) {
  // $G = \{g_1,g_2,...\}$, $F = \{f_1,f_2,...\}$ are sets of cubes
  foreach $g_i$ {
    $V^{gi} = \emptyset$
   foreach $f_j$ {
       if($f_j$ contains all literals of $g_i$) {
         $v_{ij} = f_j$ - literals of $g_i$
           $V^{gi} = V^{gi} \cup v_{ij}$
       }
    }
  }
  $H = \cap_i V^{gi}$
  $R = F - GH$
  return (H,R);
}
```

Weak Division

- Example

  $F = ace + ade + bc + bd + be + a'b + ab$
  $G = ae + b$

  $V^{ae} = c + d$
  $V^{b} = c + d + e + a' + a$

  $H = c + d \;=\; F/G$ $\qquad H = \cap V^{g_i}$
  $R = be + a'b + ab$ $\qquad R = F \setminus GH$

  $F = (ae + b)(c + d) + be + a'b + ab$

Weak Division

- We use filters to prevent trying a division
  - G is not an algebraic divisor of F if
    - G contains a literal not in F,
    - G has more terms than F,
    - For any literal, its count in G exceeds that in F, or
    - F is in the transitive fanin of G

Weak Division

- Weak_Div provides a method to divide an expression for a given divisor

- How do we find a "good" divisor?
  - Restrict to algebraic divisors
  - Generalize to Boolean divisors

- Problem:

  Given a set of functions { F_i }, find common weak (algebraic) divisors.

Divisor Identification
Primary Divisor

- Definition:

  An expression is cube-free if no cube divides the expression evenly (i.e. there is no literal that is common to all the cubes)

  "ab+c" is cube-free
  "ab+ac" and "abc" are not cube-free

  - Note: A cube-free expression must have more than one cube

- Definition:

  The primary divisors of an expression F are the set of expressions

  $D(F) = \{F/c \mid c \text{ is a cube}\}$

  Note that F/c is the quotient of a weak division

Divisor Identification
Kernel and Co-Kernel

- **Definition:**

  The kernels of an expression F are the set of expressions

  $K(F) = \{G \mid G \in D(F) \text{ and } G \text{ is cube-free}\}$

  - In other words, the kernels of an expression F are the cube-free primary divisors of F

- **Definition:**

  A cube c used to obtain the kernel $K = F/c$ is called a co-kernel of K

  - $C(F)$ is used to denote the set of co-kernels of F

Divisor Identification
Kernel and Co-Kernel

- Example

  $x = adf + aef + bdf + bef + cdf + cef + g$

  $= (a + b + c)(d + e)f + g$

kernels	co-kernels
a+b+c	df, ef
d+e	af, bf, cf
(a+b+c)(d+e)f+g	1

Divisor Identification
Kernel and Kernel Intersection

- Fundamental Theorem

  If two expressions F and G have the property that

  $\forall k_F \in K(F), \forall k_G \in K(G) \rightarrow | k_G \cap k_F | \leq 1$

  (k_G and k_F have at most one term in common),

  then F and G have no common algebraic divisors with more than one cube

  - Important:

    If we “kernel” all functions and there are no nontrivial intersections, then the only common algebraic divisors left are single cube divisors

---

Divisor Identification
Kernel Level

- Definition:

  A kernel is of level 0 (K^0) if it contains no kernels except itself

  A kernel is of level n or less (K^n) if it contains at least one kernel of level (n-1) or less, but no kernels (except itself) of level n or greater

  - $K^n(F)$ is the set of kernels of level n or less
  - $K^0(F) \subset K^1(F) \subset K^2(F) \subset \ldots \subset K^n(F) \subset K(F)$
  - level-n kernels = $K^n(F) \setminus K^{n-1}(F)$

- Example:

  $F = (a + b(c + d))(e + g)$

  $k_1 = a + b(c + d) \quad \in K^1$

  $\notin K^0$ ==> level-1

  $k_2 = c + d \quad \in K^0$

  $k_3 = e + g \quad \in K^0$

Divisor Identification
Kerneling Algorithm

```
Algorithm KERNEL(j, G) {
  R = ∅
  if(CUBE_FREE(G)) R = {G}
  for(i=j+1,...,n) {
    if($l_i$ appears only in one term)       continue
    if(∃k ≤ i, $l_k$ ∈ all cubes of G/$l_i$)     continue
    R = R ∪  KERNEL(i, MAKE_CUBE_FREE(G/$l_i$))
  }
  return R
}
MAKE_CUBE_FREE(F) removes algebraic cube factor from F
```

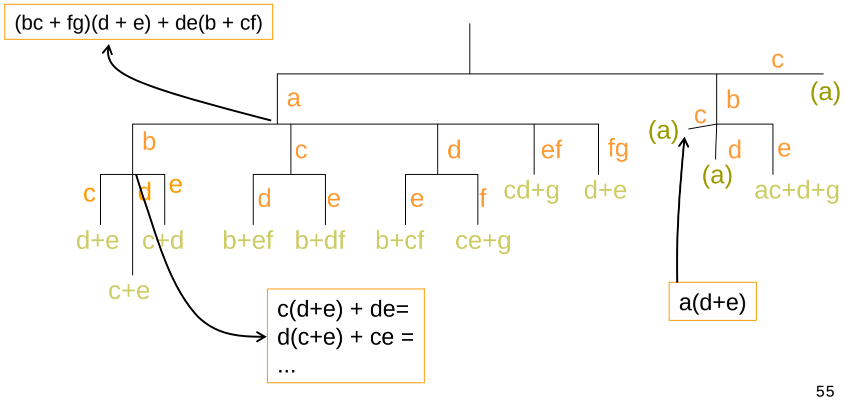
Divisor Identification
Kerneling Algorithm

- KERNEL(0, F) returns all the kernels of F
- Note:
  - The test “(∃k ≤ i, l_k ∈ all cubes of G/l_i)” in the kerneling algorithm is a major efficiency factor. It also guarantees that no co-kernel is tried more than once
  - Can be used to generate all co-kernels

Divisor Identification
Kerneling Algorithm

- Example

  F = abcd + abce + adfg + aefg + adbe + acdef + beg



Divisor Identification
Kerneling Algorithm

- Example

co-kernels	kernels
1	a((bc + fg)(d + e) + de(b + cf))) + beg
a	(bc + fg)(d + e) + de(b + cf)
ab	c(d+e) + de
abc	d + e
abd	c + e
abe	c + d
ac	b(d + e) + def
acd	b + ef

Note: F/bc = ad + ae = a(d + e)

Factor

```
Algorithm FACTOR(F) {
  if(F has no factor) return F
  // e.g. if |F|=1, or F is an OR of single literals
  // or of no literal appears more than once
  D     = CHOOSE_DIVISOR(F)
  (Q,R) = DIVIDE(F,D)
  return FACTOR(Q)xFACTOR(D) + FACTOR(R) //recur
}
```

- different heuristics can be applied for CHOOSE_DIVISOR
- different DIVIDE routines may be applied (algebraic division, Boolean division)

Factor

- Example:
  F = abc + abd + ae + af + g
  D = c + d
  Q = ab
  P = ab(c + d) + ae + af + g
  O = ab(c + d) + a(e + f) + g

Notation:
F = original function
D = divisor
Q = quotient
P = partial factored form
O = final factored form by FACTOR restricting to algebraic operations only

- Problem 1:
  O is not optimal since not maximally factored and can be further factored to “a(b(c + d) + e + f) + g”
  - It occurs when quotient Q is a single cube, and some of the literals of Q also appear in the remainder R

Factor

- To solve Problem 1
  - Check if the quotient Q is not a single cube, then done
  - Else, pick a literal l_1 in Q which occurs most frequently in cubes of F. Divide F by l_1 to obtain a new divisor D_1.
    Now, F has a new partial factored form
    $(l_1)(D_1) + (R_1)$
    and literal l_1 does not appear in R_1.
    - Note: The new divisor D_1 contains the original D as a divisor because l_1 is a literal of Q. When recursively factoring D_1, D can be discovered again.

Factor

- Example:
  F = ace + ade + bce + bde + cf + df
  D = a + b
  Q = ce + de
  P = (ce + de)(a + b) + (c + d) f
  O = e(c + d)(a + b) + (c + d)f

Notation:
F = original function
D = divisor
Q = quotient
P = partial factored form
O = final factored form by FACTOR restricting to algebraic operations only

- Problem 2:
  O is not maximally factored because “(c + d)” is common to both products “e(c + d)(a + b)” and “(c + d)f”
  - The final factored form should have been “(c+d)(e(a + b) + f)”

Factor

- To solve Problem 2
  - Essentially, we reverse D and Q!!
    - Make Q cube-free to get Q_1
    - Obtain a new divisor D_1 by dividing F by Q_1
    - If D_1 is cube-free, the partial factored form is $F = (Q_1)(D_1) + R_1$, and can recursively factor Q_1, D_1, and R_1
    - If D_1 is not cube-free, let $D_1 = cD_2$ and $D_3 = Q_1D_2$. We have the partial factoring $F = cD_3 + R_1$. Now recursively factor D_3 and R_1.

Factor

```
Algorithm GFACTOR(F, DIVISOR, DIVIDE) { // good factor
  D = DIVISOR(F)
  if(D = 0) return F
  Q = DIVIDE(F,D)
  if (|Q| = 1) return LF(F, Q, DIVISOR, DIVIDE)
  Q = MAKE_CUBE_FREE(Q)
  (D, R) = DIVIDE(F,Q)
  if (CUBE_FREE(D)) {
    Q = GFACTOR(Q, DIVISOR, DIVIDE)
    D = GFACTOR(D, DIVISOR, DIVIDE)
    R = GFACTOR(R, DIVISOR, DIVIDE)
    return Q × D + R
  }
  else {
    C = COMMON_CUBE(D) // common cube factor
    return LF(F, C, DIVISOR, DIVIDE)
  }
}
```

Factor

```
Algorithm LF(F, C, DIVISOR, DIVIDE) { // literal factor
  L = BEST_LITERAL(F, C)  //L ∈ C most frequent in F
  (Q, R) = DIVIDE(F, L)
  C = COMMON_CUBE(Q)      // largest one
  Q = CUBE_FREE(Q)
  Q = GFACTOR(Q, DIVISOR, DIVIDE)
  R = GFACTOR(R, DIVISOR, DIVIDE)
  return    L × C × Q + R
}
```

Factor

- Various kinds of factoring can be obtained by choosing different forms of DIVISOR and DIVIDE
- CHOOSE_DIVISOR:
  - LITERAL - chooses most frequent literal
  - QUICK_DIVISOR - chooses the first level-0 kernel
  - BEST_DIVISOR - chooses the best kernel
- DIVIDE:
  - Algebraic Division
  - Boolean Division

Factor

- Example

  x = ac + ad + ae + ag + bc + bd +be + bf + ce + cf + df + dg

  LITERAL_FACTOR:
  x = a(c + d + e + g) + b(c + d + e + f) + c(e + f) + d(f + g)

  QUICK_FACTOR:
  x = g(a + d) + (a + b)(c + d + e) + c(e + f) + f(b + d)

  GOOD_FACTOR:
  (c + d + e)(a + b) + f(b + c + d) + g(a + d) + ce

Factor

- QUICK_FACTOR uses GFACTOR, first level-0 kernel DIVISOR, and WEAK_DIV

- Example

  x = ae + afg + afh + bce + bcfg + bcfh + bde + bdfg + bcfh

  D = c + d ---- level-0 kernel (first found)
  Q = x/D = b(e + f(g + h)) ---- weak division
  Q = e + f(g + h) ---- make cube-free
  (D, R) = WEAK_DIV(x, Q) ---- second division
  D = a + b(c + d)
  x = QD + R R = 0

  x = (e + f(g + h)) (a + b(c + d))

Decomposition

- Decomposition is the same as factoring except:
  - divisors are added as new nodes in the network.
  - the new nodes may fan out elsewhere in the network in both positive and negative phases

```
Algorithm DECOMP($f_i$) {
  k = CHOOSE_KERNEL($f_i$)
  if (k == 0) return
  $f_{m+j}$ = k                             // create new node m + j
  $f_i$  = ($f_i$/k)$y_{m+j}$+($f_i$/k')$y'_{m+j}$+r // change node i using
                                            // new node for kernel
  DECOMP($f_i$)
  DECOMP($f_{m+j}$)
}
```

Similar to factoring, we can define
QUICK_DECOMP: pick a level 0 kernel and improve it
GOOD_DECOMP: pick the best kernel

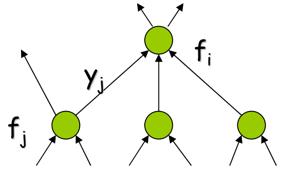
Substitution

- Idea: An existing node in a network may be a useful divisor in another node. If so, no loss in using it (unless delay is a factor).

- Algebraic substitution consists of the process of algebraically dividing the function f_i at node i in the network by the function f_j (or by f'_j) at node j. During substitution, if f_j is an algebraic divisor of f_i, then f_i is transformed into

  $f_i = qy_j + r$ (or $f_i = q_1y_j + q_0y'_j + r$)

- In practice, this is tried for each node pair of the network. n nodes in the network ⇒ $O(n^2)$ divisions.



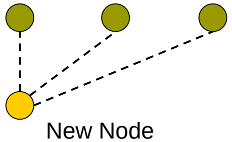
Extraction

- Recall: Extraction operation identifies common sub-expressions and restructures a Boolean network
  - Combine decomposition and substitution to provide an effective extraction algorithm

```
Algorithm EXTRACT
  foreach node n {
    DECOMP(n) // decompose all network nodes
  }
  foreach node n {
    RESUB(n)  // resubstitute using existing nodes
  }
  ELIMINATE_NODES_WITH_SMALL_VALUE
}
```

Extraction

- Kernel Extraction:
  1. Find all kernels of all functions
  2. Choose kernel intersection with best “value”
  3. Create new node with this as function
  4. Algebraically substitute new node everywhere
  5. Repeat 1,2,3,4 until best value $\leq$ threshold



New Node

Extraction

- Example

f_1 = ab(c(d + e) + f + g) + h
f_2 = ai(c(d + e) + f + j) + k
(only level-0 kernels used in this example)

1. Extraction:

$K^0(f_1) = K^0(f_2) = \{d + e\}$
$K^0(f_1) \cap K^0(f_2) = \{d + e\}$
l = d + e
f_1 = ab(cl + f + g) + h
f_2 = ai(cl + f + j) + k

2. Extraction:

$K^0(f_1) = \{cl + f + g\}; K^0(f_2) = \{cl + f + j)$
$K^0(f_1) \cap K^0(f_2)$ = cl + f
m = cl + f
f_1 = ab(m + g) + h
f_2 = ai(m + j) + k

No kernel intersections anymore!!

3. Cube extraction:

n = am
f_1 = b(n + ag) + h
f_2 = i(n + aj) + k

Extraction
Rectangle Covering

- Alternative method for extraction
- Build co-kernel cube matrix $M = R^T C$
  - rows correspond to co-kernels of individual functions
  - columns correspond to individual cubes of kernel
  - m_{ij} = cubes of functions
  - m_{ij} = 0 if cube not there
- Rectangle covering:
  - identify sub-matrix $M^* = R^{*T} C^*$, where $R^* \subseteq R$, $C^* \subseteq C$, and $m^*_{ij} \neq 0$
  - construct divisor d corresponding to M* as new node
  - extract d from all functions

Extraction
Rectangle Covering

- Example

  F = af + bf + ag + cg + ade + bde + cde
  G = af + bf + ace + bce
  H = ade + cde

  Kernels/Co-kernels:
  F: (de+f+g)/a
  (de + f)/b
  (a+b+c)/de
  (a + b)/f
  (de+g)/c
  (a+c)/g
  G: (ce+f)/{a,b}
  (a+b)/{f,ce}
  H: (a+c)/de

$$M = \begin{array}{cc|ccccccc} & & a & b & c & ce & de & f & g \\ \hline F & a & & & & & ade & af & ag \\ F & b & & & & & bde & bf & \\ F & de & ade & bde & cde & & & & \\ F & f & af & bf & & & & & \\ F & c & & & & & cde & & cg \\ F & g & ag & & cg & & & & \\ G & a & & & & ace & & af & \\ G & b & & & & bce & & bf & \\ G & ce & ace & bce & & & & & \\ G & f & af & bf & & & & & \\ H & de & ade & & cde & & & & \end{array}$$

Extraction
Rectangle Covering

- Example (cont'd)

  F = af + bf + ag + cg + ade + bde + cde
  G = af + bf + ace + bce
  H = ade + cde

  - Pick sub-matrix M'
  - Extract new expression X

    F = fx + ag + cg + dex + cde
    G = fx + cex
    H =ade + cde
    X = a + b
  - Update M

$$M = \begin{array}{cc|ccccccc} & & a & b & c & ce & de & f & g \\ \hline F & a & & & & & ade & af & ag \\ F & b & & & & & bde & bf & \\ F & de & \boxed{ade} & \boxed{bde} & cde & & & & \\ F & f & \boxed{af} & \boxed{bf} & & & & & \\ F & c & & & & & cde & & cg \\ F & g & ag & & cg & & & & \\ G & a & & & & ace & & af & \\ G & b & & & & bce & & bf & \\ G & ce & \boxed{ace} & \boxed{bce} & & & & & \\ G & f & \boxed{af} & \boxed{bf} & & & & & \\ H & de & ade & & cde & & & & \end{array}$$

Extraction
Rectangle Covering

- Number literals before - Number of literals after

$$V(R',C') = \sum_{i\in R,\, j\in C} v_{ij} - \sum_{i\in R'} w_i^r - \sum_{j\in C} w_j^c$$

v_{ij} : Number of literals of cube m_{ij}

w_i^r: (Number of literals of the cube associated with row i)+1

w_j^c: Number of literals of the cube associated with column j

- For prior example
  - V = 20 - 10 - 2 = 8

$$M = \begin{array}{cc|ccccccc} & & a & b & c & ce & de & f & g \\ \hline F & a & & & & & ade & af & ag \\ F & b & & & & & bde & bf & \\ F & de & \boxed{ade} & \boxed{bde} & cde & & & & \\ F & f & \boxed{af} & \boxed{bf} & & & & & \\ F & c & & & & & cde & & cg \\ F & g & ag & & cg & & & & \\ G & a & & & & ace & & af & \\ G & b & & & & bce & & bf & \\ G & ce & \boxed{ace} & \boxed{bce} & & & & & \\ G & f & \boxed{af} & \boxed{bf} & & & & & \\ H & de & ade & & cde & & & & \end{array}$$

Extraction
Rectangle Covering

- Pseudo Boolean Division
  - Idea: consider entries in covering matrix that are don't cares
    - overlap of rectangles (a+a = a)
    - product that cancel each other out (a·a' = 0)
  - Example:

    F = ab' + ac' + a'b + a'c + bc' + b'

$$M = \begin{array}{cc|cccccc} & & a & b & c & a' & b' & c' \\ \hline F & a & & & & \boxed{*} & \boxed{ab'} & \boxed{ac'} \\ F & b & & & & \boxed{a'b} & \boxed{*} & \boxed{bc'} \\ F & c & & & & \boxed{a'c} & \boxed{b'c} & \boxed{*} \\ F & a' & * & a'b & a'c & & & \\ F & b' & ab' & * & b'c & & & \\ F & c' & ac' & bc' & * & & & \end{array}$$

Result:
X = a' + b' + c'
F = ax + bx + cx

Fast Kernel Computation

- Non-robustness of kernel extraction
  - Recomputation of kernels after every substitution: expensive
  - Some functions may have many kernels (e.g. symmetric functions)
- Cannot measure if kernel can be used as complemented node
- Solution: compute only subset of kernels:
  - Two-cube “kernel” extraction [Rajski et al ‘90]
  - Objects:
    - 2-cube divisors
    - 2-literal cube divisors
  - Example: f = abd + a’b’d + a’cd
    - ab + a’b’, b’ + c and ab + a’c are 2-cube divisors.
    - a’d is a 2-literal cube divisor.

Fast Kernel Computation

- Properties of fast divisor (kernel) extraction:
  - $O(n^2)$ number of 2-cube divisors in an n-cube Boolean expression
  - Concurrent extraction of 2-cube divisors and 2-literal cube divisors
  - Handle divisor and complemented divisor simultaneously

- Example:

f = abd + a’b’d + a’cd

k = ab + a’b’, k’ = ab’ + a’b (both 2-cube divisors)

j = ab + a’c, j’ = ab’ + a’c’ (both 2-cube divisors)

c = ab (2-literal cube), c’ = a’ + b’ (2-cube divisor)

Fast Kernel Computation

- Generating all two cube divisors

  $F = \{c_i\}$

  $D(F) = \{d \mid d = \text{make\_cube\_free}(c_i + c_j)\}$
  - c_i, c_j are any pair of cubes of cubes in F
    - I.e., take all pairs of cubes in F and makes them cube-free
  - Divisor generation is $O(n^2)$, where n = number of cubes in F

- Example:

  F = axe + ag + bcxe + bcg

  make_cube_free(c_i + c_j) = {xe + g, a + bc, axe + bcg, ag + bcxe}
  - Note: Function F is made into an algebraic expression before generating double-cube divisors
  - Not all 2-cube divisors are kernels (why?)

Fast Kernel Computation

- Key results of 2-cube divisors

Theorem: Expressions F and G have a common multiple-cube divisors if and only if $D(F) \cap D(G) \neq 0$

Proof:

If:

If $D(F) \cap D(G) \neq 0$ then $\exists d \in D(F) \cap D(G)$ which is a double-cube divisor of F and G. d is a multiple-cube divisor of F and of G.

Only if:

Suppose $C = \{c_1, c_2, \ldots, c_m\}$ is a multiple-cube divisor of F and of G. Take any $e = (c_i + c_j)$. If e is cube-free, then $e \in D(F) \cap D(G)$. If e is not cube-free, then let d = make_cube_free(c_i + c_j). d has 2 cubes since F and G are algebraic expressions. Hence $d \in D(F) \cap D(G)$.

Fast Kernel Computation

- Example:

  Suppose that $C = ab + ac + f$ is a multiple divisor of F and G

  If $e = ac + f$, e is cube-free and $e \in D(F) \cap D(G)$

  If $e = ab + ac$, $d = \{b + c\} \in D(F) \cap D(G)$

  As a result of the Theorem, all multiple-cube divisors can be “discovered” by using just double-cube divisors

---

Fast Kernel Computation

- Algorithm:
  - Generate and store all 2-cube kernels (2-literal cubes) and recognize complement divisors
  - Find the best 2-cube kernel or 2-literal cube divisor at each stage and extract it
  - Update 2-cube divisor (2-literal cubes) set after extraction
  - Iterate extraction of divisors until no more improvement
- Results:
  - Much faster
  - Quality as good as that of kernel extraction

Boolean Division

- What’s wrong with algebraic division?
  - Divisor and quotient are orthogonal!
  - Better factored form might be:

    $(g_1 + g_2 + \ldots + g_n)\ (d_1 + d_2 + \ldots + d_m)$
    - g_i and d_j may share same literals
    - redundant product literals
      - Example

        $abe+ace+abd+cd \ / \ (ae+d) = \varnothing$

        But: $aabe+ace+abd+cd \ / \ (ae+d) = (ab+c)$
    - g_i and d_j may share opposite literals
    - product terms are non-existing
      - Example

        $a'b+ac+bc \ / \ (a'+c) = \varnothing$

        But: $a'a+a'b+ac+bc \ / \ (a'+c) = (a+b)$

Boolean Division

- Definition:

  g is a Boolean divisor of f if h and r exist such that $f = gh + r$, $gh \neq 0$

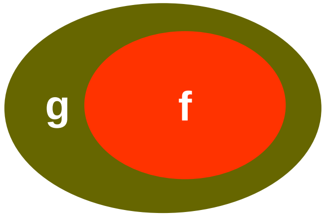
  g is said to be a factor of f if, in addition, $r = 0$, i.e., $f = gh$

  - h is called the quotient
  - r is called the remainder
  - h and r may not be unique

Boolean Division

- **Theorem:**

  A logic function g is a Boolean factor of a logic function f if and only if f ⊆ g (i.e. fg' = 0, i.e. g' ⊆ f')



Boolean Division

Proof:

(⇒) g is a Boolean factor of f. Then ∃h such that f = gh; Hence, f ⊆ g (as well as h).

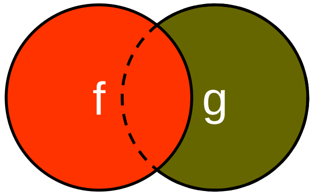
(⇐) f ⊆ g ⇒ f = gf = g(f + r) = gh. (Here r is any function r ⊆ g'.)

- Note:
  - h = f works fine for the proof
  - Given f and g, h is not unique
  - To get a small h is the same as to get a small f + r. Since rg = 0, this is the same as minimizing (simplifying) f with DC = g'.

Boolean Division

- **Theorem:**

  g is a Boolean divisor of f if and only if fg ≠ 0



Boolean Division

Proof:

(⇒) f = gh + r, gh ≠ 0 ⇒ fg = gh + gr. Since gh ≠ 0, fg ≠ 0.

(⇐) Assume that fg ≠ 0. f = fg + fg' = g(f + k) + fg'. (Here k ⊆ g'.)
Then f = gh + r, with h = f + k, r = fg'. Since gh = fg ≠ 0, then gh ≠ 0.

- Note:
  - f has many divisors. We are looking for some g such that f = gh+r, where g, h, r are simple functions. (simplify f with DC = g')

Boolean Division
Incomplete Specified Function

- $F = (f,d,r)$
- Definition:

  A completely specified logic function g is a Boolean divisor of F if there exist h, e (completely specified) such that

  $$f \subseteq gh + e \subseteq f + d$$

  and $gh \not\subseteq d$.
- Definition:

  g is a Boolean factor of F if there exists h such that

  $$f \subseteq gh \subseteq f + d$$

Boolean Division
Incomplete Specified Function

- Lemma:

  $f \subseteq g$ if and only if g is a Boolean factor of F.

  Proof:

  (⇒) Assume that $f \subseteq g$. Let $h = f + k$ where $kg \subseteq d$.

  Then $hg = (f + k)\, g \subseteq (f + d)$.

  Since $f \subseteq g$, $fg = f$ and thus $f \subseteq (f + k)\, g = gh$.

  Thus

  $$f \subseteq (f + k)\, g \subseteq f + d$$

  (⇐) Assume that $f = gh$.

  Suppose ∃ minterm m such that $f(m) = 1$ but $g(m) = 0$.

  Then $f(m) = 1$ but $g(m)h(m) = 0$ implying that $f \not\subseteq gh$.

  Thus $f(m) = 1$ implies $g(m) = 1$, i.e. $f \subseteq g$
- Note:
  - Since $kg \subseteq d$, $k \subseteq (d + g')$. Hence obtain $h = f + k$ by simplifying f with DC = $(d + g')$.

Boolean Division
Incomplete Specified Function

- Lemma:

  $fg \neq 0$ if and only if g is a Boolean divisor of F.

  Proof:

  (⇒) Assume $fg \neq 0$.

  Let $fg \subseteq h \subseteq (f + d + g')$ and $fg' \subseteq e \subseteq (f + d)$.

  Then $f = fg + fg' \subseteq gh + e \subseteq g(f + d + g') + f + d = f + d$

  Also, $0 \neq fg \subseteq gh \rightarrow ghf \neq 0$.

  Now $gh \not\subseteq d$, since otherwise $ghf = 0$ (since $fd = 0$), verifying the conditions of Boolean division.

  (⇐) Assume that g is a Boolean divisor.

  Then ∃h such that $gh \not\subseteq d$ and

  $f \subseteq gh + e \subseteq f + d$

  Since $gh = (ghf + ghd) \not\subseteq d$, then $fgh \neq 0$ implying that $fg \neq 0$.

Boolean Division
Incomplete Specified Function

- Recipe for Boolean division:

  $(f \subseteq gh + e \subseteq f + d)$
  - Choose g such that $fg \neq 0$
  - Simplify fg with DC = $(d + g')$ to get h
  - Simplify fg' with DC = $(d + fg)$ to get e (could use DC = $d + gh$)
- $fg \subseteq h \subseteq f + d + g'$

  $fg' \subseteq e \subseteq fg' + d + fg = f + d$

Boolean Division

- Given *F* = (f,d,r), write a cover for *F* in the form *gh* + *e* where h and e are minimal in some sense

Algorithm:

1. Create a new variable x to “represent” g
2. Form the don’t care set ($\tilde{d}$ = xg’ + x’g)
   (Since x = g we don’t care if x ≠ g)
3. Minimize (f $\tilde{d}'$, d + $\tilde{d}$, r $\tilde{d}'$) to get $\tilde{f}$
4. Return (h = $\tilde{f}$/x, e) where e is the remainder of $\tilde{f}$
   (These are simply the terms not containing x)
5. f/x denote weak algebraic division

Boolean Division

- Note that (f $\tilde{d}'$, d + $\tilde{d}$, r $\tilde{d}'$) is a partition. We can use ESPRESSO to minimize it, but the objective there is to minimize the number of cubes - not completely appropriate.

- Example:
  f = a + bc
  g = a + b

  $\tilde{d}$ = xa’b’ + x’(a+b) where x = g = (a+b)

  - Minimize (a + bc) $\tilde{d}'$ = (a + bc) (x’a’b’ + x(a+b)) = xa + xbc
    with DC = xa’b’ + x ’(a+b)
  - A minimum cover is a + bc but it does not use x or x’ !
  - Force x in the cover. This yields f = a + xc = a + (a+b) c.

  Heuristic:
  Find answer with x in it and which also uses the least variables (or literals)

Boolean Division

Assume F is a cover for $\Im$ = (f,d,r) and D is a cover for d.

First Algorithm:

```
Algorithm Boolean_Divide1(F,D,G) {
  $D_1 = D + xG' + x'G$                  // (don't care)
  $F_1 = FD_1'$                          // (care on-set)
  $R_1 = (F_1 + D_1)' = F_1'D_1' = F'D_1'$   // (care off-set)
  $F_2$ = remove x' from $F_1$           // positive substitution only
  $F_3$ = MIN_LITERAL($F_2$, $R_1$, x) // Filter for Espresso
            // (minimum literal support including x)
  $F_4$ = ESPRESSO($F_3$,$D_1$,$R_1$)
  H = $F_4$/x                            // (quotient)
  E = $F_4$ - {xH}                       // (remainder)
  return (HG+E)
}
```

Boolean Division

Assume F is a cover for $\Im$ = (f,d,r) and D is a cover for d.

Second Algorithm:

```
Algorithm Boolean_Divide2(F,D,G) {
  $D_1 = D + xG' + x'G$                      // (don't care)
  $F_1 = FD_1'$                              // (on-set)
  $R_1 = (F_1 + D_1)' = F_1'D_1' = F'D_1'$   // (off-set)
  // $F_2$ = remove x' from $F_1$  (difference to first alg.)
  $F_3$ = MIN_LITERAL($F_2$, $R_1$, x, x') // Filter for Espresso
                // (minimum literal support including x)
  $F_4$ = ESPRESSO($F_3$,$D_1$,$R_1$)
  $H_1 = F_4/x$                              // (first quotient)
  $H_0 = F_4/x'$                             // (first quotient)
  $E = F_4 - (\{xH_1\}+\{x'H_0\})$           // (remainder)
  return ($GH_1+G'H_0+E$)
}
```

Boolean Division
Minimal Literal Support

- Support minimization (MINVAR)

  Given:
  $\Im$ = (f,d,r)
  F = $\{c^1, c^2, \ldots., c^k\}$ (a cover of $\Im$)
  R = $\{r^1, r^2, \ldots, r^m\}$ (a cover of r)

  1. Construct blocking matrix B^i for each c^i
  2. Form “super” blocking matrix B
  3. Find a minimum cover S of B,
     S = $\{ j_1, j_2, \ldots, j_v \}$.
  4. Modify $\tilde{F} \leftarrow \{ \tilde{c}^1, \tilde{c}^2, \ldots, \tilde{c}^k \}$ where

$$B = \begin{bmatrix} B^1 \\ B^2 \\ \vdots \\ B^k \end{bmatrix}$$

$$(\tilde{c}^i)_j = \begin{cases} (\tilde{c}^i)_j & \text{if } j \in S \\ \{0,1\} = 2 & \text{otherwise} \end{cases}$$

Boolean Division
Minimal Literal Support

- Given:
  $\Im$ = (f,d,r)
  F = $\{c^1, c^2, \ldots., c^k\}$ (a cover of $\Im$)
  R = $\{r^1, r^2, \ldots, r^m\}$ (a cover of r)
  n: number of variables
  Literal Blocking Matrix:

$$(\hat{B}^i)_{q,j} = \begin{cases} 1 \text{ if } v_j \in c^i \text{ and } v'_j \in r^q \\ 0 \text{ otherwise} \end{cases}$$

$$(\hat{B}^i)_{q,j+n} = \begin{cases} 1 \text{ if } v'_j \in c^i \text{ and } v_j \in r^q \\ 0 \text{ otherwise} \end{cases}$$

- Example:
  c^i = ad’e’, r^q = a’ce

$$\hat{B}^i_q = \begin{array}{cccccccccc} a & b & c & d & e & a' & b' & c' & d' & e' \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Boolean Division
Minimal Literal Support

- Example (literal blocking matrix)
  on-set cube: c^i = ab’d
  off-set: r = a’b’d’ + abd’ + acd’ + bcd + c’d’

	a	b	c	d	a’	b’	c’	d’
a’b’d’	1	0	0	1	0	0	0	0
abd’	0	0	0	1	0	1	0	0
acd’	0	0	0	1	0	0	0	0
bcd	0	0	0	0	0	1	0	0
c’d’	0	0	0	1	0	0	0	0

- Minimum column cover {d,b’}
- Thus b’d is the maximum prime covering ab’d

- Note:
  For one cube, minimum literal support is the same as minimum variable support

---

Boolean Division

- Example
  F = a + bc
  Algebraic division: F/(a + b) = 0
  Boolean division: F ÷ (a + b) = a + c

  1. Let x = a + b
  2. Generate don’t care set: D_1 = x’(a + b) + xa’b’.
  3. Generate care on-set:
     - $F_1 = F \cap D_1'$ = (a + bc)(xa + xb +x’a’b’) =ax + bcx.
     - Let *C* = $\{c^1$ = ax, c^2 = bcx$\}$
  4. Generate care off-set:
     - $R_1 = F'D_1'$ = (a’b’ + a’c’)(xa + xb + x’a’b’) =a’bc’x + a’b’x’.
     - Let R = $\{r^1$ = a’bc’x, r^2 = a’b’x’$\}$.
  5. Form super-variable blocking matrix using column order (a, b, c, x), with a’,b’,c’,x’ omitted.

$$B = \begin{bmatrix} B^1 \\ B^2 \end{bmatrix} = \begin{bmatrix} 1000 \\ 1001 \\ 0010 \\ 0101 \end{bmatrix}$$

Boolean Division

- Example (cont'd)

6. Find minimum column cover = {a, c, x}
7. Eliminate in F_1 all variables associated with b
   So F_1 = ax + bcx = ax + cx = x(a + c)
8. Simplifying (applying expand, irredundant on F_1), we get F_1 = a + xc
9. Thus quotient = F_1/x = c, remainder = a
10. F = a + bc = a + cx = a + c(a + b)

It is important that x is forced in the cover!

$$B = \begin{bmatrix} B^1 \\ B^2 \end{bmatrix} = \begin{array}{c} \begin{array}{cccc} a & b & c & x \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$