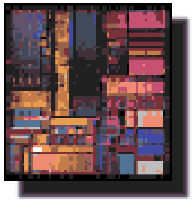
ECE 595Z
Digital VLSI Design Automation

Module 5 (Lectures 14-20): Multi-level Synthesis
Lecture 16



Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

Lectures 14-15: Recap

- Boolean Network
  - Like a network of gates, except each internal node is an SOP expression
- Multi-level synthesis : Iterative improvement (transformations on network)
- Algebraic Division
  - Treat Boolean expressions as polynomials in real numbers and divide
- Kernels / Co-kernels
  - Useful in generating common divisors (intersection of kernels, intersection of co-kernels)
- Recursive algorithm for kernel generation

Kernel Generation Algorithm

- Recursive algorithm – call it on any kernels that you find to discover additional kernels
- Processes variables in lexicographic order

```
KERNELS(j, G) {
  if (G is cube-free)
    R = {G};
  else R = {};
  for (i = j+1, …, n) {
    if (l_i appears in more than one cube) {
      if (∃k ≤ i, l_k ∈ all cubes of G/l_i )
        continue;
      else {
        R = R ∪ KERNELS(i, cube_free(G/l_i))
      }
    }
  }
return R;
}
```

A cube-free function is a trivial kernel for itself

Process remaining variables in lexicographic order

Speedup technique

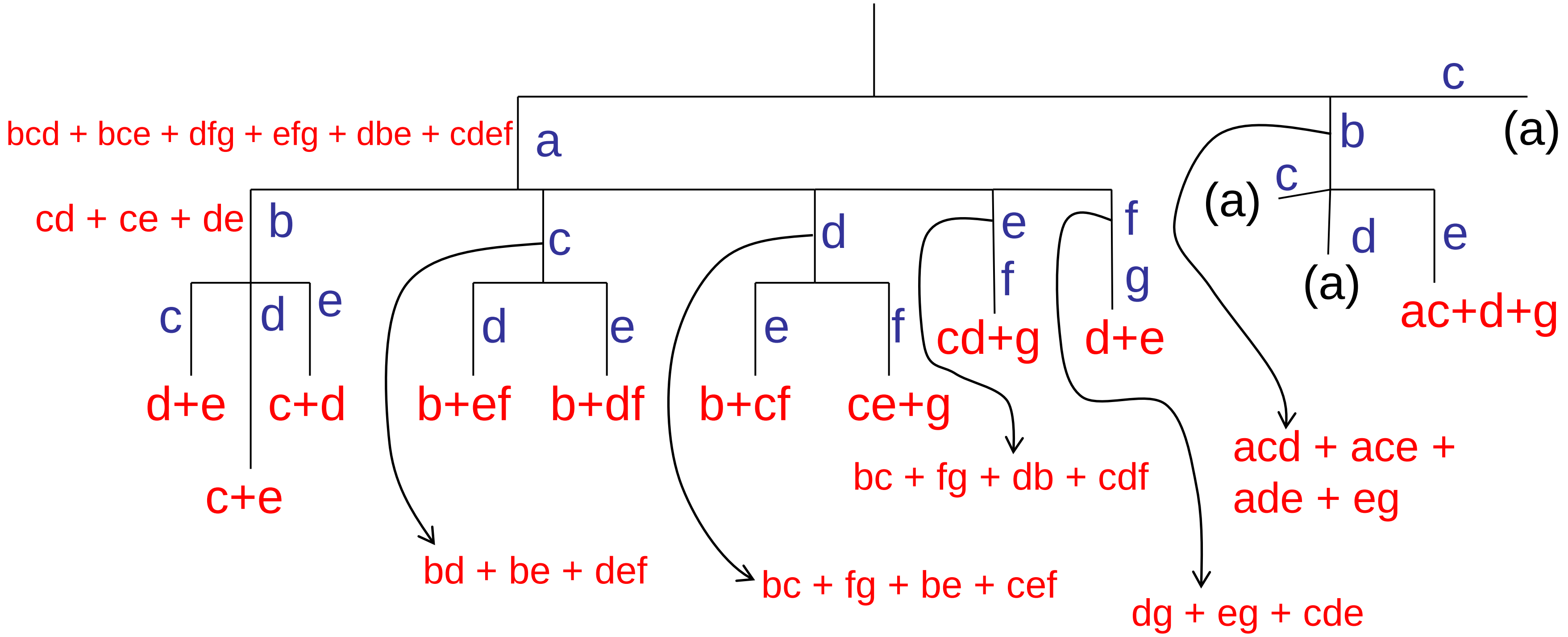
Recursive call

KERNELS(0,f) returns all kernels of f

Kernel Generation Illustrated

- Another Example
  - Literals along the path represent the co-kernel

$$z = abcd + abce + adfg + aefg + adbe + acdef + beg$$



Kernel Generation Illustrated

$z = abcd + abce + adfg + aefg + adbe + acdef + beg$

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	$b + ef$
....	……..

Finding Common Factors Using Kernels

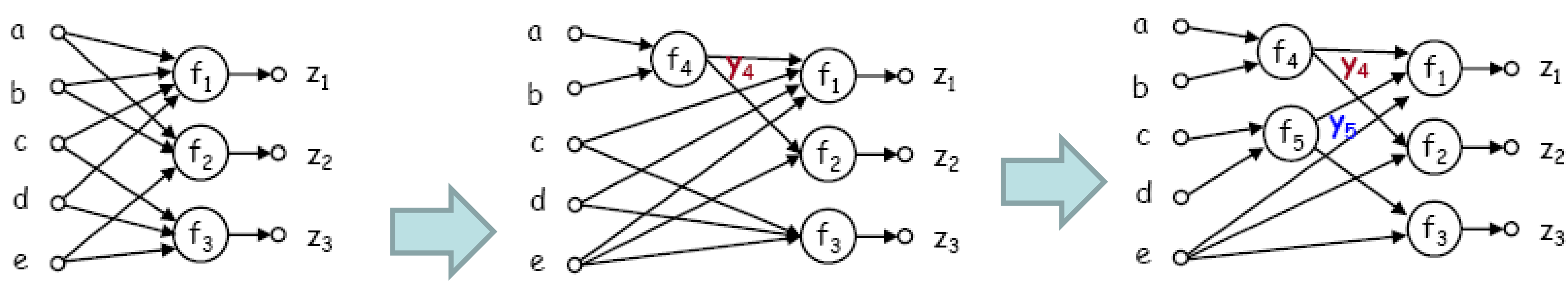
- Extraction of a multi-cube common factor
  - Intersection of kernels
- Extraction of a single cube common factor
  - Intersection of co-kernels!

Efficient Techniques for Extraction of Single-cube and Multi-cube Factors

R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, “MIS: a multiple-level logic optimization system,” IEEE Tr. on CAD, Feb. 1990.

Extraction

- Given a Boolean network, identify sub-expressions common to two or more nodes, which can be used to reduce the #literals



$f_1 = acd + bcd + e$
$f_2 = ae' + be' + de'$
$f_3 = cde$

$f_1 = y_4cd + e$
$f_2 = y_4e' + de'$
$f_3 = cde$
$f_4 = a + b$

Literals saved = __

$f_1 = y_4y_5 + e$
$f_2 = y_4e' + de'$
$f_3 = y_5e$
$f_4 = a + b$
$f_5 = cd$

Literals saved = __

Set Operations on 0-1 Matrices

- General idea: 0-1 matrices form efficient data structures for performing set operations
- Let $\{S_1, S_2, \ldots S_m\}$ be sets on elements $e_1, \ldots, e_n$
- Construct a matrix where each set is a row and each element is a column.

Example:

	e_1	e_2	e_3	e_4	e_5
S_1	1	0	0	1	1
S_2	1	1	1	0	1
S_3	1	1	1	0	0

$S_1 = \{e_1, e_4, e_5\}$
$S_2 = \{e_1, e_2, e_3, e_5\}$
$S_3 = \{e_1, e_2, e_3\}$

Set Operations on 0-1 Matrices

- A **rectangle** (R, C) is a sub-matrix with all 1's, i.e., a sub-set of rows R and a sub-set of columns C such that

  $$M(i,j) = 1 \ \forall \ i \in R, j \in C$$

- Note: Rows and columns need not be contiguous
- What do rectangles mean?
  – Elements common to the sets (set intersections)
- A **prime rectangle** is not contained in any other rectangle

Example:

	e_1	e_2	e_3	e_4	e_5
S_1	1	0	0	1	1
S_2	1	1	1	0	1
S_3	1	1	1	0	0

R = {2,3}, C = {1,2,3}

$S_2 \cap S_3 = \{e_1, e_2, e_3\}$

R = {1,2}, C = {1,5}

$S_1 \cap S_2 = \{e_1, e_5\}$

Applications to Extracting Factors

- Identifying kernel / co-kernel pairs for an expression
- Single-cube extraction
- Multi-cube extraction

Identifying Co-kernel/Kernel Pairs

- Given a single SOP expression
- **Cube-literal matrix** : Matrix with one row for each cube, and one column for each literal
  - View cubes as sets of literals
- A prime rectangle with at least two rows in the cube-literal matrix indicates a kernel/co-kernel pair

Example:

$f = x_1x_2x_3x_4x_7 + x_1x_2x_3x_4x_8 + x_1x_2x_3x_5 + x_1x_2x_3x_6 + x_1x_2x_9$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
$x_1x_2x_3x_4x_7$	1	1	1	1	0	0	1	0	0
$x_1x_2x_3x_4x_8$	1	1	1	1	0	0	0	1	0
$x_1x_2x_3x_5$	1	1	1	0	1	0	0	0	0
$x_1x_2x_3x_6$	1	1	1	0	0	1	0	0	0
$x_1x_2x_9$	1	1	0	0	0	0	0	0	1

R = {1,2}, C = {1,2,3,4}
Co-kernel =
Kernel =

R = {1,2,3,4}, C = {1,2,3}
Co-kernel =
Kernel =

Extracting Single-cube Factors

- Given a set of SOP expressions
- Construct cube-literal matrix with cubes and literals from ALL expressions
  - Rows : cubes
  - Columns : literals
- Rectangles in the matrix = single cube factors
- Prime rectangles = maximal single-cube factors

Example:

$f_1 = x_1x_2x_3 + x_4x_5$
$f_2 = x_1x_2x_3 + x_4x_6 + x_6$
$f_3 = x_1x_2x_4$

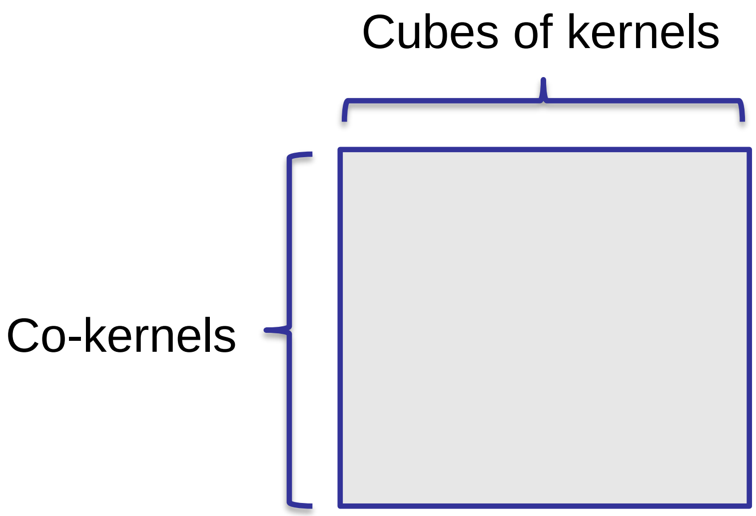
		x_1	x_2	x_3	x_4	x_5	x_6
f_1	$x_1x_2x_3$	1	1	1	0	0	0
	x_4x_5	0	0	0	1	1	0
f_2	$x_1x_2x_3$	1	1	1	0	0	0
	x_4x_6	0	0	0	1	0	1
	x_6	0	0	0	0	0	1
f_3	$x_1x_2x_4$	1	1	0	1	0	0

R = {1,3}, C = {1,2,3}
Factor =

R = {1,3,6}, C = {1,2,}
Factor =

Extracting Multi-cube Factors

- Given two or more SOP expressions
- Compute set of kernels /co-kernels for each expression
- Construct **co-kernel cube matrix**: Matrix with rows corresponding to co-kernels, columns corresponding to unique kernel cubes
- Rectangles in the matrix = kernel intersections = multi-cube factors



Extracting Multi-cube Factors

Note: identical co-kernels from different functions → different rows
Identical kernel cubes → only 1 column

- 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}

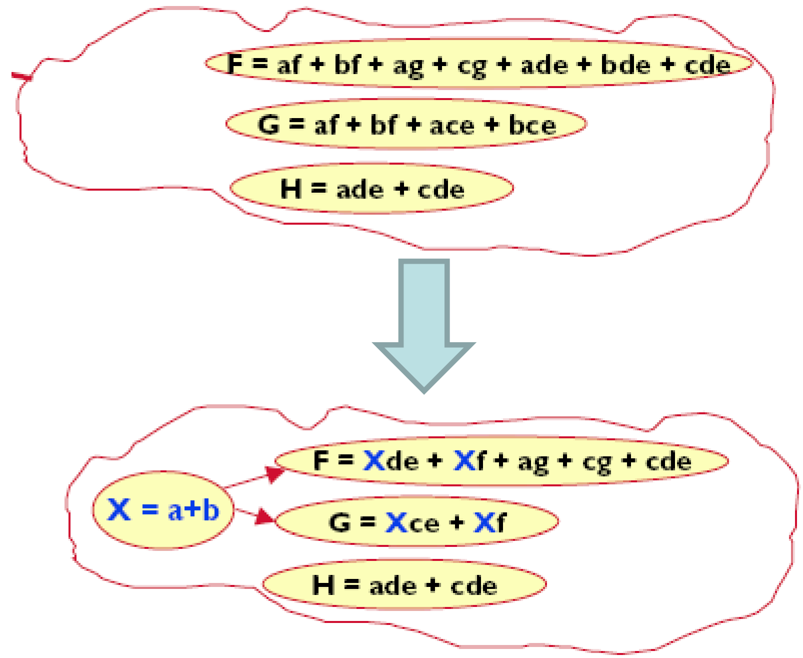
H: (a+c)/de

			a 1	b 2	c 3	ce 4	de 5	f 6	g 7
F	a	1	.	.	.	.	1	1	1
F	b	2	.	.	.	.	1	1	.
F	de	3	1	1	1	.	.	.	.
F	f	4	1	1	.	.	.	.	.
F	c	5	.	.	.	.	1	.	1
F	g	6	1	.	1	.	.	.	.
G	a	7	.	.	.	1	.	1	.
G	b	8	.	.	.	1	.	1	.
G	ce	9	1	1	.	.	.	.	.
G	f	10	1	1	.	.	.	.	.
H	de	11	1	.	1	.	.	.	.

Extracting Multi-cube Factors

Example:

			a 1	b 2	c 3	ce 4	de 5	f 6	g 7
F	a	1	.	.	.	.	1	1	1
F	b	2	.	.	.	.	1	1	.
F	de	3	1	1	1	.	.	.	.
F	f	4	1	1	.	.	.	.	.
F	c	5	.	.	.	.	1	.	1
F	g	6	1	.	1	.	.	.	.
G	a	7	.	.	.	1	.	1	.
G	b	8	.	.	.	1	.	1	.
G	ce	9	1	1	.	.	.	.	.
G	f	10	1	1	.	.	.	.	.
H	de	11	1	.	1	.	.	.	.



- Question : How does this relate to the Brayton-McMullen theorem on multi-cube factors?

Extracting Factors Using Rectangle Generation / Covering

- General Idea : Iteratively find a rectangle in the cube-literal and co-kernel-cube matrices
  - Alternative : Pick several rectangles at a time (covering, like ESPRESSO)
- Issues
  - Which factor to pick at any point?
  - What to do once you pick a factor?

Selecting Factors

- Use literal count to guide the selection of factors
  - How many literals would be saved if this factor is extracted as a new node in the network and substituted in all applicable nodes?
- **Value** of a prime rectangle
  - Estimate of improvement in the overall Boolean Logic Network by factoring out the term corresponding to a rectangle
  - **How can we compute this easily from the rectangle itslef?**

Weight of Rectangles: Single-cube Case

- Consider a cube-literal matrix (used to identify single-cube factors)
- Define: the **weight** of a row of the cube-lit matrix
  - It's just = 1 for each row
- Define: the **weight** of a column of cube-lit matrix
  - It's just = 1 for each column
- Define: the **weight** of a rectangle of a cube-lit matrix
  - If just 1 row in rectangle (R,C): $\Sigma_{c \text{ in } C}$ (weight of column c)
  - Else: $\Sigma_{c \text{ in } C}$ (weight of column c) + $\Sigma_{r \text{ in } R}$ (weight of row r)

Example:

	x_1	x_2	x_3	x_4	x_5	x_6
$x_1x_2x_3$	1	1	1	0	0	0
x_4x_5	0	0	0	1	1	0
$x_1x_2x_3$	1	1	1	0	0	0
x_4x_6	0	0	0	1	0	1
x_6	0	0	0	0	0	1
$x_1x_2x_4$	1	1	0	1	0	0

Weight of R={1,3} C={1,2,3}:

Weight of R={1,3,6} C={1,2}:

Value of Rectangles: Single-cube Case

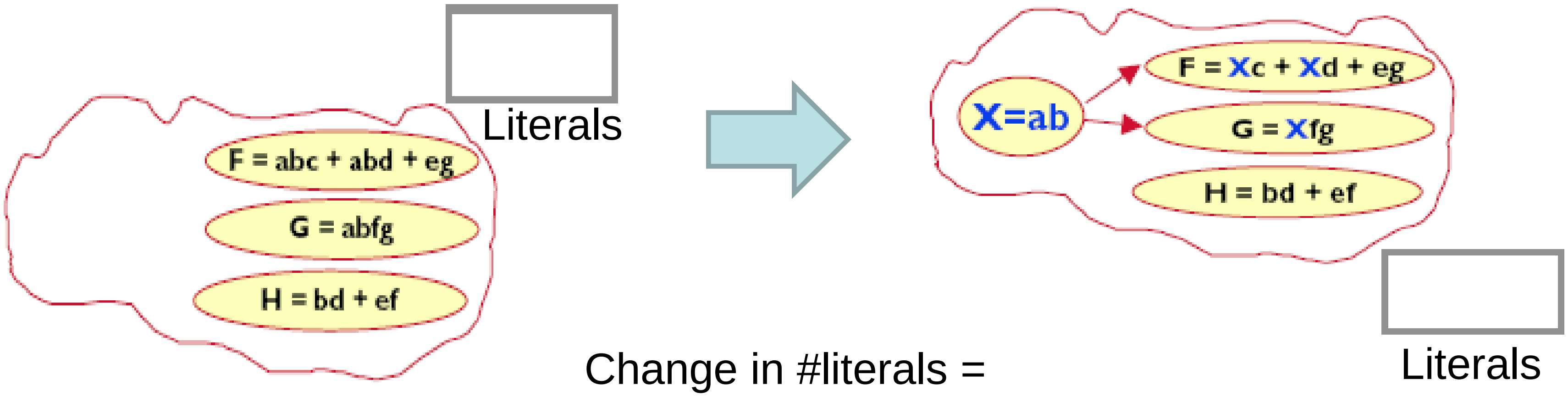
- Define: the **value** of an element of cube-lit matrix
  - It's = 1 if there is a "1" in that entry, else it's = 0
- Define: the **value** of a rectangle of a cube-lit matrix
  - [$\Sigma_{r \text{ in } R} \Sigma_{c \text{ in } C}$ (value of element r,c in matrix)] - (weight of rectangle)
- In words, the value of a rectangle is:
  - (# of "1"s covered by the rectangle) - (# of rows + # cols of the rectangle)

	a 1	b 2	c 3	d 4	e 5	f 6	g 7
abc 1	1	1	1	.	.	.	.
abd 2	1	1	.	1	.	.	.
eg 3	.	.	.	.	1	.	1
abfg 4	1	1	.	.	.	1	1
bd 5	.	1	.	1	.	.	.
ef 6	.	.	.	.	1	1	.

Weight =
Value =

Why is "Value" Useful?

- Example



		a 1	b 2	c 3	d 4	e 5	f 6	g 7
abc	1	1	1	1	.	.	.	.
abd	2	1	1	.	1	.	.	.
eg	3	.	.	.	.	1	.	1
abfg	4	1	1	.	.	.	1	1
bd	5	.	1	.	1	.	.	.
ef	6	.	.	.	.	1	1	.

#1s covered by the rect =
#rows + #cols of rect =
value = (#1s) - (#rows+#cols) =

Value of Rectangles: Multi-cube Case

- Define: **weight** of column of co-kernel cube matrix
  - The number of literals in the cube that labels the column
- Define: **weight** of row of co-kernel cube matrix
  - 1 + number of literals in the co-kernel that labels the row
- Define: **value** of an element of the co-kernel cube matrix
  - If the matrix has a "1" : number of literals in the product term you get by ANDing row cube label with column cube label
  - If matrix has a "0" : 0

		a	*b*	*c*	*ce*	*d*	*f*	*g*
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*				

Value of Rectangles: Multi-cube Case

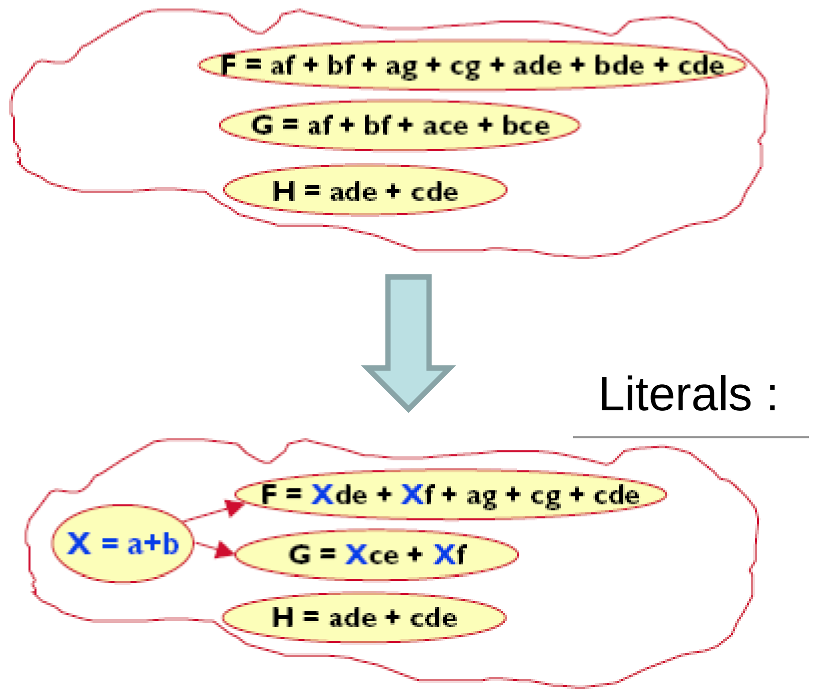
- Weight of a rectangle
  - [Σ (row weights) + Σ (col weights)]
- Value of a rectangle
  - [Σ element value of each "1" covered] - [Σ (row weights) + Σ (col weights)]
- Same interpretation
  - Value tells you how many literals are saved after you extract the factor that the rectangle represents

Value of Rectangles : Multi-cube Case

- Example

			a 1	b 2	c 3	ce 4	de 5	f 6	g 7
F	a	1	.	.	.	.	1	1	1
F	b	2	.	.	.	.	1	1	.
F	de	3	1	1	1			.	.
F	f	4	1	1	.			.	.
F	c	5			.			.	1
F	g	6	1	.	1	.	.	.	.
G	a	7	.	.	.	1	.	1	.
G	b	8	.	.	.			1	.
G	ce	9	1	1	.			.	.
G	f	10	1	1	.			.	.
H	de	11	1		1	.	.	.	.

Literals :



Value of rectangle:
Change in # literals :