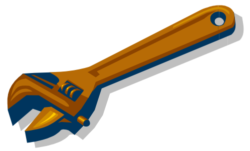
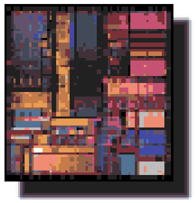


# ECE 595Z
# Digital VLSI Design Automation

## Module 5 (Lectures 14-20): Multi-level Synthesis
## Lecture 17



Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

## Lecture #16: Summary

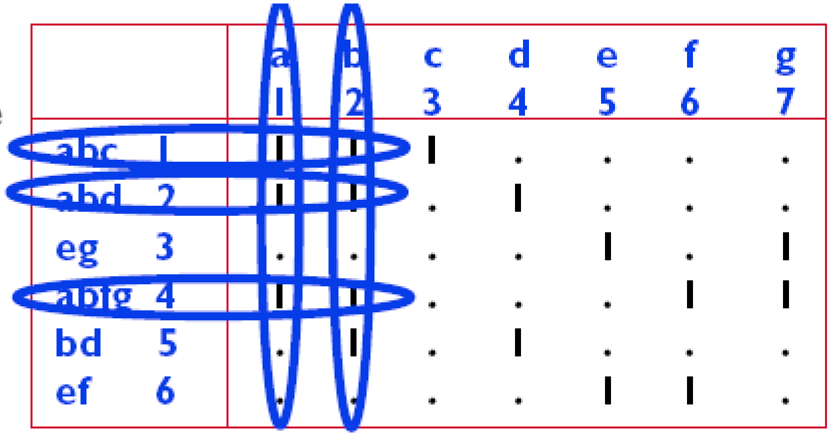
- 0-1 Matrices can be used to represent sets
  - Prime rectangles represent set intersections
- Applications to extraction
  - Co-kernels / kernels are prime rectangles in **cube-literal matrix** for a *single* expression
  - Single cube factors are prime rectangles in cube-literal matrix for *multiple* expressions
  - Multi-cube factors are prime rectangles in **co-kernel-cube matrix**
- How to select the “best” factor?
  - Value of rectangles represent change in #literals

## Using "Value" to Select Factors

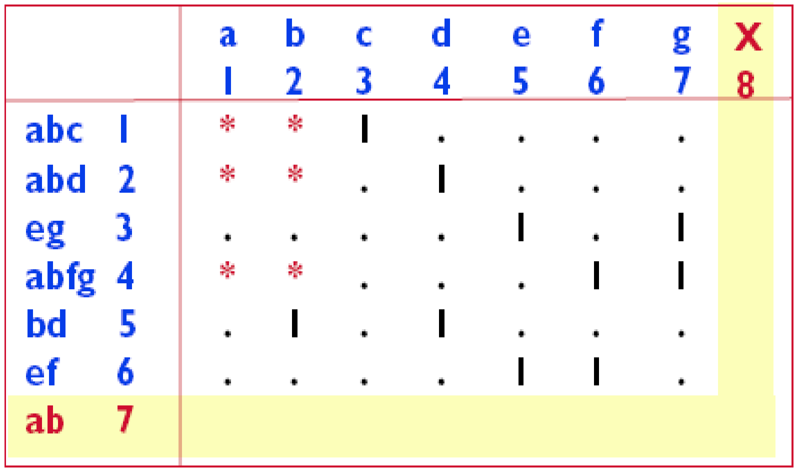
- Value is really a measure of change in #literals if we select a given factor!
- Heuristic approach to factoring:
  1. Find a rectangle of maximal value
  2. Extract it
  3. Update matrix
  – Repeat Steps 1-3 with the next best rectangle

# Matrix Update: Single-cube Case

- Once you extract a factor, need to update the cube-literal matrix, since the network has changed
  - Add a column at end of matrix, label it
  - Add another row at bottom of matrix, label it
  - Change all the “1” entries of the prime rectangle we just extracted into “*” to indicate don’t cares
    - Why? It’s OK to cover these guys again with the next rectangle we extract, but if you don’t cover them it’s OK too



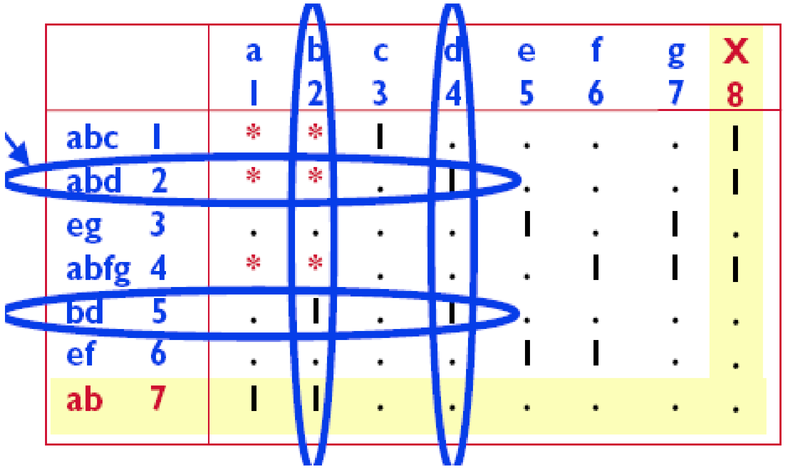
| | | a 1 | b 2 | c 3 | d 4 | e 5 | f 6 | g 7 |
|---|---|---|---|---|---|---|---|---|
| abc | 1 | 1 | 1 | 1 | . | . | . | . |
| abd | 2 | 1 | 1 | . | 1 | . | . | . |
| eg | 3 | . | . | . | . | 1 | . | 1 |
| abfg | 4 | 1 | 1 | . | . | . | 1 | 1 |
| bd | 5 | . | 1 | . | 1 | . | . | . |
| ef | 6 | . | .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



| | | a 1 | b 2 | c 3 | d 4 | e 5 | f 6 | g 7 | X 8 |
|---|---|---|---|---|---|---|---|---|---|
| abc | 1 | * | * | 1 | . | . | . | . | |
| abd | 2 | * | * | . | 1 | . | . | . | |
| eg | 3 | . | . | . | . | 1 | . | 1 | |
| abfg | 4 | * | * | . | . | . | 1 | 1 | |
| bd | 5 | . | 1 | . | 1 | . | . | . | |
| ef | 6 | . | . | . | . | 1 | 1 | . | |
| ab | 7 | | | | | | | | |

# Side-effects of Rectangle Overlap: Single-cube Case

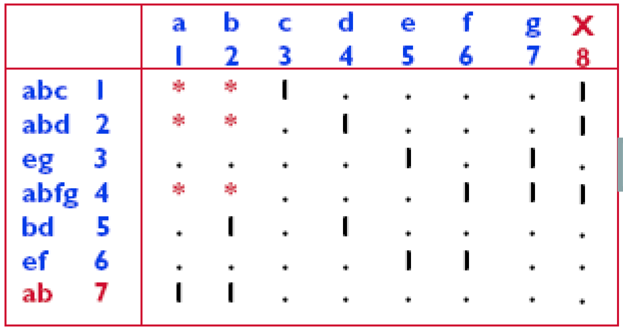
- It is OK for a rectangle to cover some "*" entries
- Redefine "rectangle" to mean "rows, cols don't cover any 0 entries", *i.e.*, it's OK to cover the "*" entries or don't cares
- However, this "messes up" an assumption of the algebraic model!



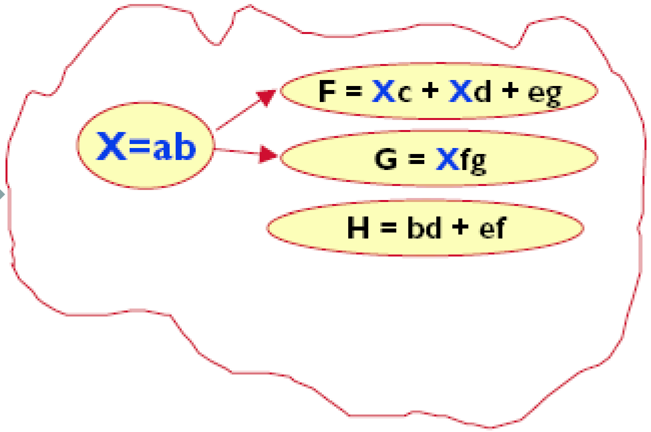
| | a 1 | b 2 | c 3 | d 4 | e 5 | f 6 | g 7 | X 8 |
|---|---|---|---|---|---|---|---|---|
| abc 1 | * | * | 1 | . | . | . | . | 1 |
| abd 2 | * | * | . | 1 | . | . | . | 1 |
| eg 3 | . | . | . | . | 1 | . | 1 | . |
| abfg 4 | * | * | . | . | . | 1 | 1 | 1 |
| bd 5 | . | 1 | . | 1 | . | . | . | . |
| ef 6 | . | . | . | . | 1 | 1 | . | . |
| ab 7 | 1 | 1 | . | . | . | . | . | . |

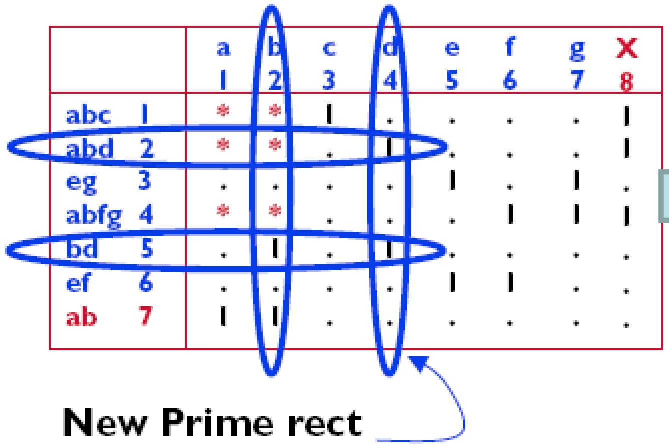
# Side-effects of Rectangle Overlap

- Example:



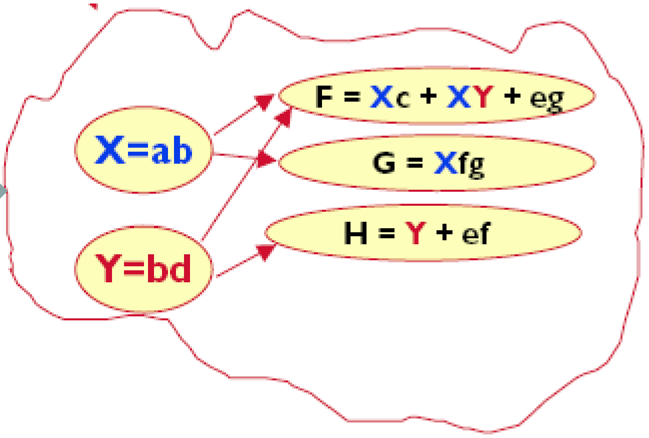
| | a 1 | b 2 | c 3 | d 4 | e 5 | f 6 | g 7 | X 8 |
|---|---|---|---|---|---|---|---|---|
| abc 1 | * | * | 1 | . | . | . | . | 1 |
| abd 2 | * | * | . | 1 | . | . | . | 1 |
| eg 3 | . | . | . | . | 1 | . | 1 | . |
| abfg 4 | * | * | . | . | . | 1 | 1 | 1 |
| bd 5 | . | 1 | . | 1 | . | . | . | . |
| ef 6 | . | . | . | . | 1 | 1 | . | . |
| ab 7 | 1 | 1 | . | . | . | . | . | . |





| | a 1 | b 2 | c 3 | d 4 | e 5 | f 6 | g 7 | X 8 |
|---|---|---|---|---|---|---|---|---|
| abc 1 | * | * | 1 | . | . | . | . | 1 |
| abd 2 | * | * | . | 1 | . | . | . | 1 |
| eg 3 | . | . | . | . | 1 | . | 1 | . |
| abfg 4 | * | * | . | . | . | 1 | 1 | 1 |
| bd 5 | . | 1 | . | 1 | . | . | . | . |
| ef 6 | . | . | . | . | 1 | 1 | . | . |
| ab 7 | 1 | 1 | . | . | . | . | . | . |

New Prime rect



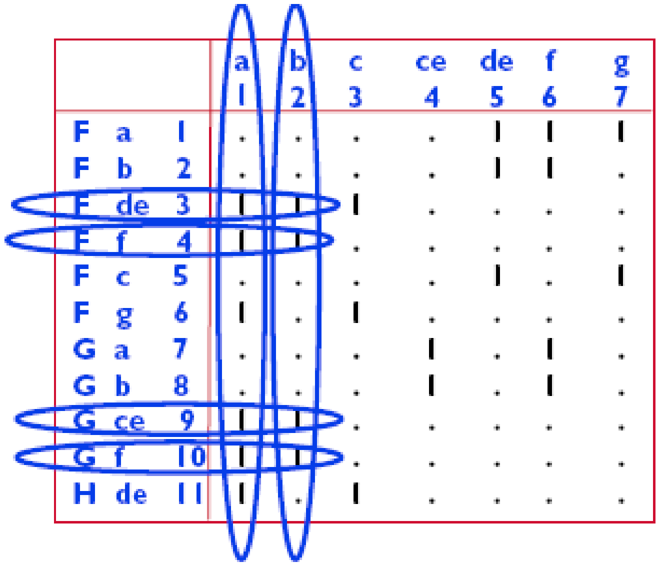
F = (ab)c + (ab)(bd) + eg
is what we *really are implementing!*

# Side-effects of Rectangle Overlap

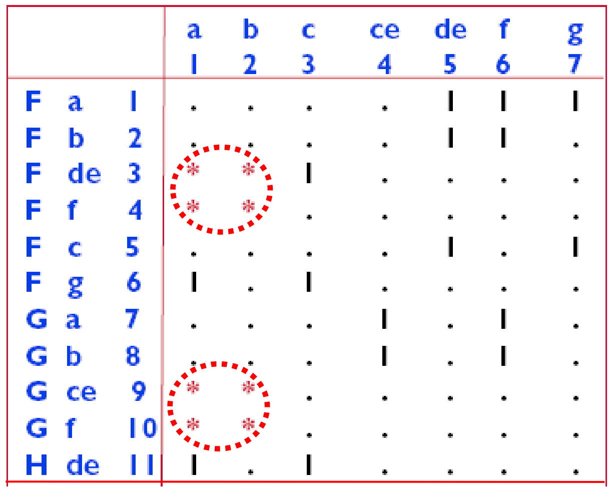
- Effect: Literals get repeated in the factoring of products
  - This is a technical violation of the algebraic model, which said that if we factor $f = d.q + r$, $d$ and $q$ should have no common variables
- Overlapping rectangles mean $d$, $q$ do have common variables
- Bottomline: Rectangle based algorithm gives you (algebraic factoring)++
  - Actually a good thing!

# Matrix Update (Multi-cube Factors)

- Consider co-kernel-cube matrix used for multi-cube extraction
- What do you do after extracting a factor corresponding to a prime rectangle?
  1. Change all the “1”s in the prime rectangle into “*”s - since they are already “covered” by the factor, they become don’t cares



| | | | a<br>1 | b<br>2 | c<br>3 | ce<br>4 | de<br>5 | f<br>6 | g<br>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 | . | . | . | . |

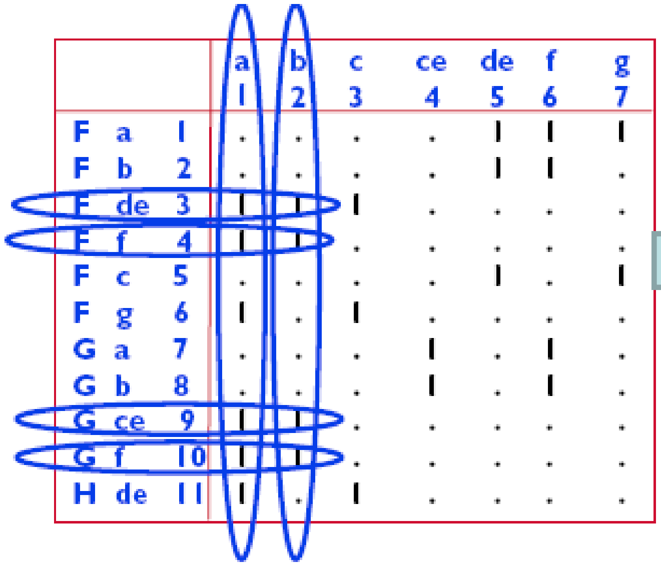


| | | | a<br>1 | b<br>2 | c<br>3 | ce<br>4 | de<br>5 | f<br>6 | g<br>7 |
|---|---|---|---|---|---|---|---|---|---|
| F | a | 1 | . | . | . | . | 1 | 1 | 1 |
| F | b | 2 | . | . | . | . | 1 | 1 | . |
| F | de | 3 | * | * | 1 | . | . | . | . |
| F | f | 4 | * | * | . | . | . | . | . |
| F | c | 5 | . | . | . | . | 1 | . | 1 |
| F | g | 6 | 1 | . | 1 | . | . | . | . |
| G | a | 7 | . | . | . | 1 | . | 1 | . |
| G | b | 8 | . | . | . | 1 | . | 1 | . |
| G | ce | 9 | * | * | . | . | . | . | . |
| G | f | 10 | * | * | . | . | . | . | . |
| H | de | 11 | 1 | . | 1 | . | . | . | . |

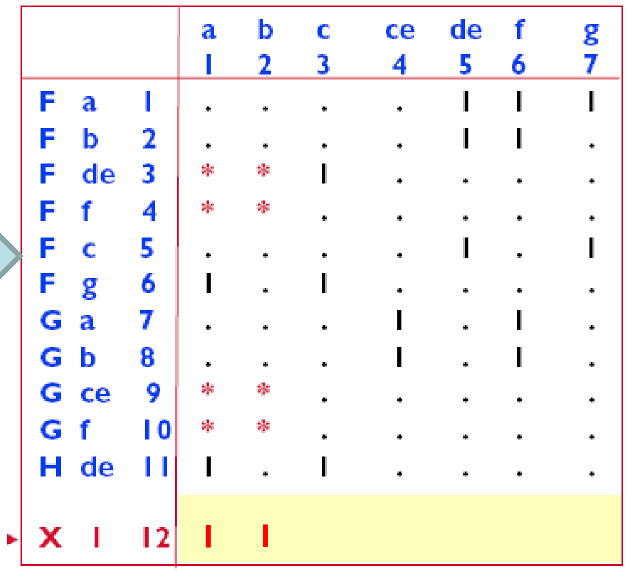
# Matrix Update (Multi-cube Factors)

- What do you do after extracting a factor corresponding to a prime rectangle?
  1. Change all the "1"s in the prime rectangle into "*"s
  2. Compute kernels for the multi-cube factor
     - For each kernel you found here, add a new row (and label as before)
     - No new columns – the factor is just an intersection of kernels found before, so no new kernel cubes

Example:



| | | | a<br>1 | b<br>2 | c<br>3 | ce<br>4 | de<br>5 | f<br>6 | g<br>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 | . | . | . | . |



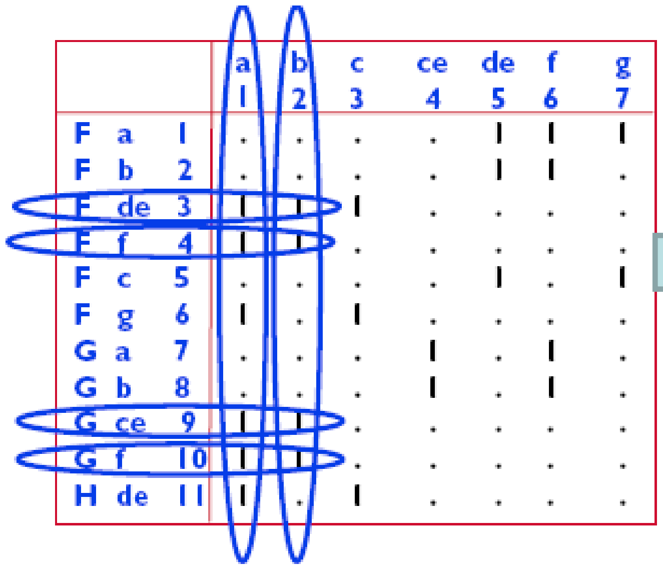
| | | | a<br>1 | b<br>2 | c<br>3 | ce<br>4 | de<br>5 | f<br>6 | g<br>7 |
|---|---|---|---|---|---|---|---|---|---|
| F | a | 1 | . | . | . | . | 1 | 1 | 1 |
| F | b | 2 | . | . | . | . | 1 | 1 | . |
| F | de | 3 | * | * | 1 | . | . | . | . |
| F | f | 4 | * | * | . | . | . | . | . |
| F | c | 5 | . | . | . | . | 1 | . | 1 |
| F | g | 6 | 1 | . | 1 | . | . | . | . |
| G | a | 7 | . | . | . | 1 | . | 1 | . |
| G | b | 8 | . | . | . | 1 | . | 1 | . |
| G | ce | 9 | * | * | . | . | . | . | . |
| G | f | 10 | * | * | . | . | . | . | . |
| H | de | 11 | 1 | . | 1 | . | . | . | . |
| X | 1 | 12 | 1 | 1 | | | | | |

X = a + b
(only 1 kernel – itself)

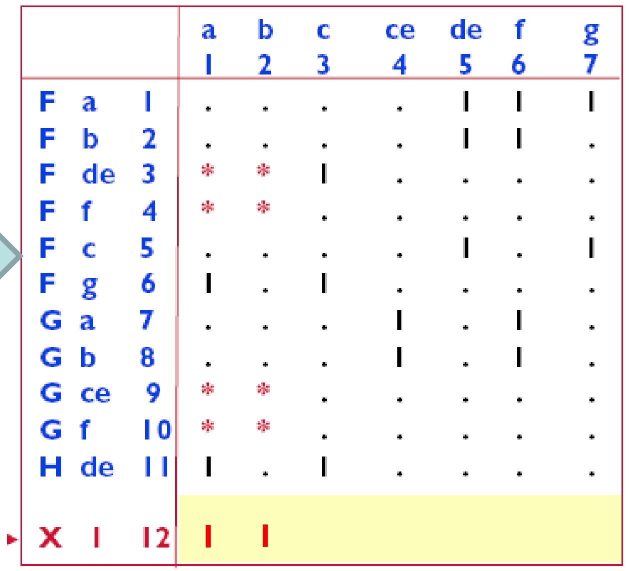
# Matrix Update (Multi-cube Factors)

- What do you do after extracting a factor corresponding to a prime rectangle?
  1. Change all the “1”s in the prime rectangle into “*”s
  2. Compute kernels for the multi-cube factor
     - For each kernel you found here, add a new row (and label as before)
     - No new columns – the factor is just an intersection of kernels found before, so no new kernel cubes
  3. **Other entries outside rectangle also become “*”s (WHY?)**

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 | . | . | . | . |



| | | | 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 | . | . | . | . |
| F | f | 4 | * | * | . | . | . | . | . |
| F | c | 5 | . | . | . | . | 1 | . | 1 |
| F | g | 6 | 1 | . | 1 | . | . | . | . |
| G | a | 7 | . | . | . | 1 | . | 1 | . |
| G | b | 8 | . | . | . | 1 | . | 1 | . |
| G | ce | 9 | * | * | . | . | . | . | . |
| G | f | 10 | * | * | . | . | . | . | . |
| H | de | 11 | 1 | . | 1 | . | . | . | . |
| X | 1 | 12 | 1 | 1 | | | | | |

F = af + bf + ag + cg + ade + bde + cde

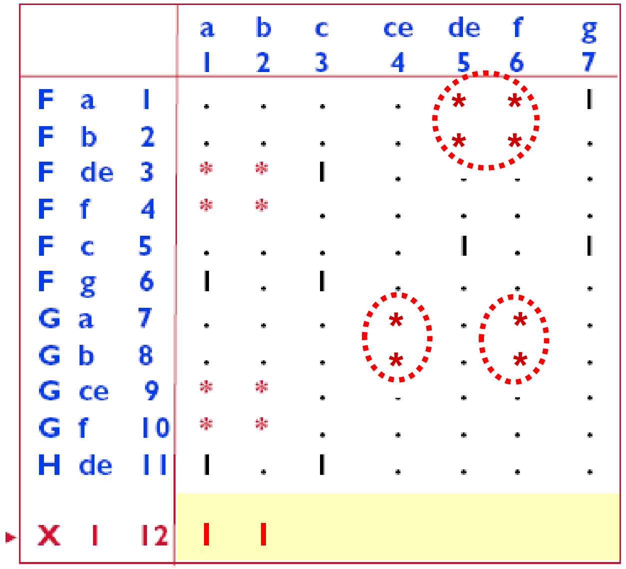
F = deX + fX + ag + cg + cde

G = af + bf + ace + bce

G = ceX + fX

# Matrix Update (Multi-cube Factors)

- Once some terms in the original SOP expressions have been "covered" by a factor, no need to consider them any more
- How can we reflect this in the co-kernel-cube matrix?
- Each "1" entry in the matrix corresponds to a product term in one of the original SOP expressions



|  |  |  | a 1 | b 2 | c 3 | ce 4 | de 5 | f 6 | g 7 |
|---|---|---|---|---|---|---|---|---|---|
| F | a | 1 | . | . | . | . | * | * | 1 |
| F | b | 2 | . | . | . | . | * | * | . |
| F | de | 3 | * | * | 1 | . | . | . | . |
| F | f | 4 | * | * | . | . | . | . | . |
| F | c | 5 | . | . | . | . | 1 | . | 1 |
| F | g | 6 | 1 | . | 1 | . | . | . | . |
| G | a | 7 | . | . | . | * | . | * | . |
| G | b | 8 | . | . | . | * | . | * | . |
| G | ce | 9 | * | * | . | . | . | . | . |
| G | f | 10 | * | * | . | . | . | . | . |
| H | de | 11 | 1 | . | 1 | . | . | . | . |
| X | 1 | 12 | 1 | 1 |  |  |  |  |  |

F = af + bf + ag + cg + ade + bde + cde

G = af + bf + ace + bce

F = deX + fX + ag + cg + cde

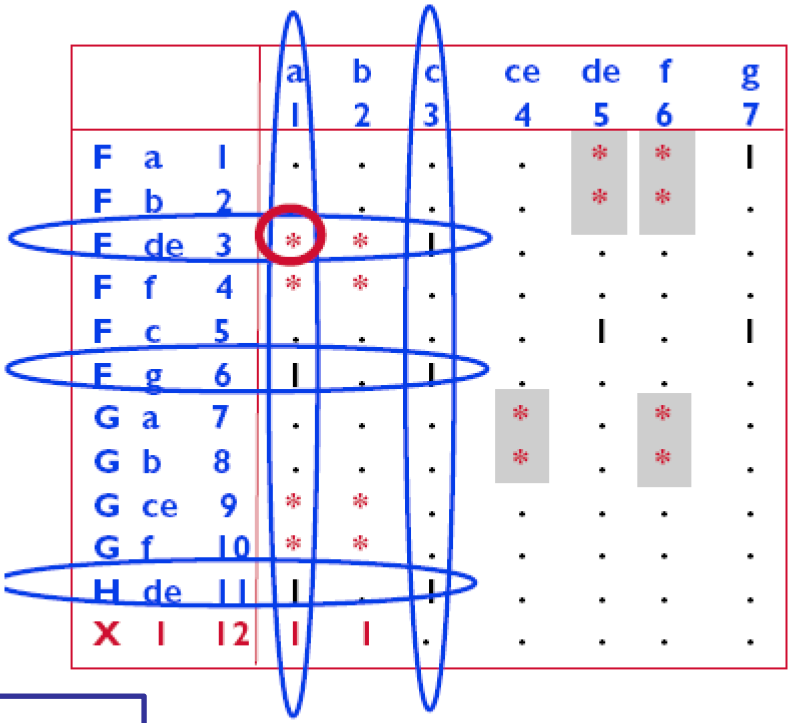
G = ceX + fX

(X = a + b)

For each product term that is covered by the extracted factor, change all entries corresponding to that term to "*"

# Side-effects of Rectangle Overlap (Multi-cube Factors)

- As with the single-cube case, it is OK for a rectangle to include “*” entries
- Once again, this breaks an assumption of the algebraic model



| | | | a | b | c | ce | de | f | g |
|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| F | a | 1 | . | . | . | . | * | * | 1 |
| F | b | 2 | | . | . | . | * | * | . |
| F | de | 3 | * | * | 1 | . | . | . | . |
| F | f | 4 | * | * | . | . | . | . | . |
| F | c | 5 | . | . | . | . | 1 | . | 1 |
| F | g | 6 | 1 | . | 1 | . | . | . | . |
| G | a | 7 | . | . | . | * | . | * | . |
| G | b | 8 | . | . | . | * | . | * | . |
| G | ce | 9 | * | * | . | . | . | . | . |
| G | f | 10 | * | * | . | . | . | . | . |
| H | de | 11 | 1 | . | 1 | . | . | . | . |
| X | 1 | 12 | 1 | 1 | . | . | . | . | . |

After 1st factor (X = a+b):
F = deX + fX + ag + cg + cde

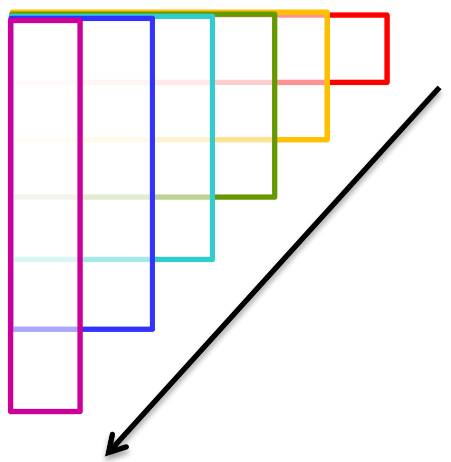
After 2nd factor (Y = a+c):
F = deX + fX + deY + gY
 = de(a+b) + f(a+b) + de(a+c) + g(a+c)
 = ade + bde + af + bf + ade + cde + ag + cg

# Re-cap: Extraction of Factors

- Algebraic model
- Kernels / Co-kernels
- Unifying theme: Rectangles in 0-1 matrices
- Single-cube
  - Cube-literal matrix
  - Prime rectangle is a good factor
- Multi-cube
  - Co-kernel-cube matrix
  - Prime rectangle is a good factor
- Rectangle weights & values estimate literals saved
- Overall approach: Iteratively generate "good" rectangles (another covering problem!)

# Algorithm for Rectangle Generation : Greedy Row

- Objective: Find prime rectangles one at a time
    1. Start with a "seed" row (rectangle with 1 row and highest value)
    2. Add a row such that resulting rectangle is of highest value (over all possible row additions).
    3. Repeat until rectangle is a single column
    4. Select best rectangle seen



**Dual algorithm - Greedy Column**: Start with seed column, grow by adding columns

## Algorithm for Rectangle Generation : Ping-Pong

- Objective: Find prime rectangles one at a time
  1. Start with a “seed” row (rectangle with 1 row and highest value)
  2. Add a row such that resulting rectangle is of highest value (over all possible row additions). Keep adding until value increases.
  3. Add a column such that resulting rectangle is of highest value. Keep adding until value increases.
  4. Go to step 2.
  5. Stop when you rectangle cannot be “profitably” grown in any direction.

Used in MIS & SIS

## Boolean Optimization

## Can we go beyond the Algebraic Model?

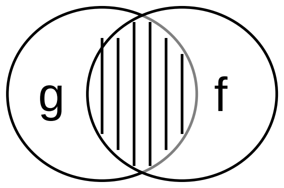
- The algebraic model enables efficient (**fast**) transformations of a Boolean network
  - Collapsing/elimination, extraction/decomposition, substitution, …
- However, it is limited in the **scope of optimization** since it does not take advantage of the unique properties of Boolean algebra.

# Boolean Division

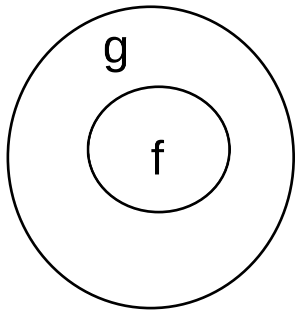
- p is a Boolean divisor of f if $q \neq 0$ and r exist such that

  $$f = pq + r$$

  - q and r are not unique
  - If $r = 0$, q is an exact factor of f
- **Theorem**: If $f.g \neq 0$, then g is a Boolean divisor of f
- **Theorem**: A logic function g is an exact Boolean factor of a logic function of f iff $f \subseteq g$
- Boolean division with a given factor is OK, but no efficient method for identifying Boolean factors is known



f = gq + r



Number of Boolean divisors and factors is LARGE!

# Boolean Division: An indirect approach

- Apply two-level minimization!
- Given $f = (f^{ON}, f^{DC}, f^{OFF})$, and $g$, find the "best" $h,r$ such that $gh + r$ is a cover for $f$.
  - Create new variable $y$ to represent output of $g$
  - Add $y \neq g$ to the don't care set for $f$
    - $f^{DC*} = yg' + y'g$
  - Minimize $(f^{ON}(f^{DC*})' , f^{DC} + f^{DC*} , f^{OFF}(f^{DC*})' )$
    - Use ESPRESSO (modify cost-function to literals)
    - Force y to appear in the result
    - Product terms with y form the quotient, other terms form the remainder

Example: $f = a + bc$, $g = a+b$

$f^{DC*} = ya'b' + y'a + y'b$

$$\begin{tikzcd} & a + bc \\ \text{Minimize } (a + bc)(f^{DC*})' \text{ with don't care set } f^{DC*} \arrow[ur] \arrow[r] & \boxed{a + yc} \end{tikzcd}$$

## Boolean Optimization Using Don't Cares

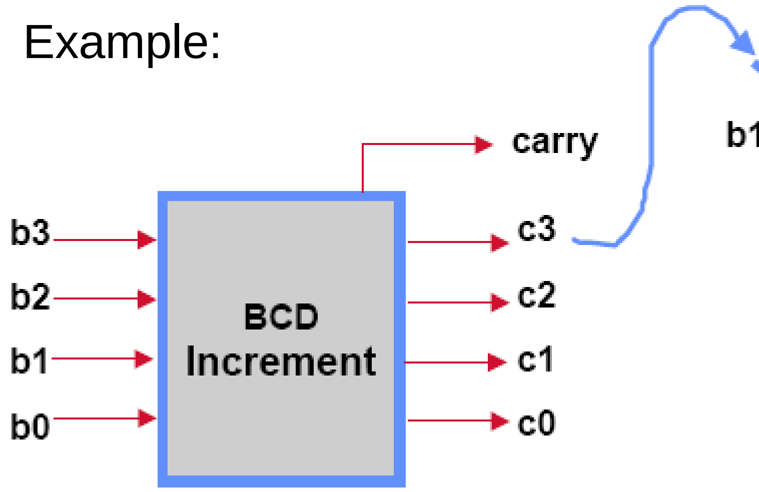
## Augmenting Algebraic Methods with Boolean Methods

- General strategy for technology-independent optimization
  - Circuit re-structuring using algebraic methods
  - Circuit simplification using Boolean methods

## Don't Cares

- From undergraduate digital logic design: Don’t cares are input patterns that could “never happen”

Example:



| b1b0 \ b3b2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | d |
| 01 | | | | d |
| 11 | | 1 | d | d |
| 10 | | | d | d |

Patterns b3 b2 b1 b0 = 1010, 1011, 1100, 1101, 1110, 1111 cannot happen

# Don't Cares in Multi-level Boolean Networks

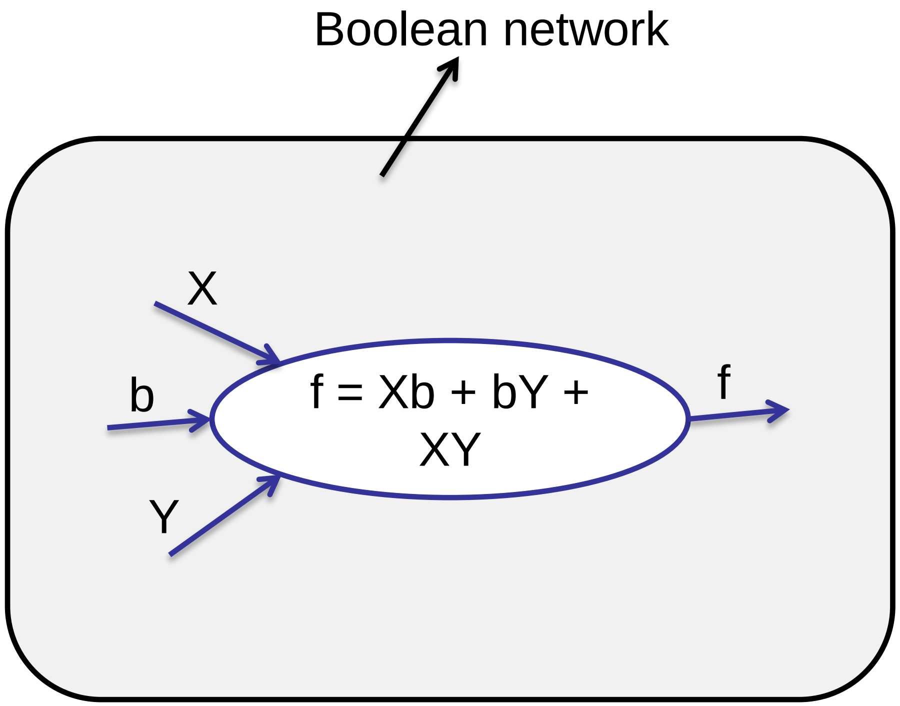
- The “usual” don’t cares
  - Somebody tells you based on the semantics of the primary inputs
  - **Explicit** don’t cares
- Don’t cares that occur due to the **network structure**
  - What we will talk about in this class
  - **Implicit** don’t cares

## Outline

- Informal introduction to implicit don’t cares in Boolean networks
- A more rigorous definition
- Optimizing Boolean networks using implicit don’t cares
- Prime and irredundant Boolean networks

# Informal Introduction to DCs

- Given a node in a Boolean network
- Can we say anything about don't cares for node f?



| Y \ Xb | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |

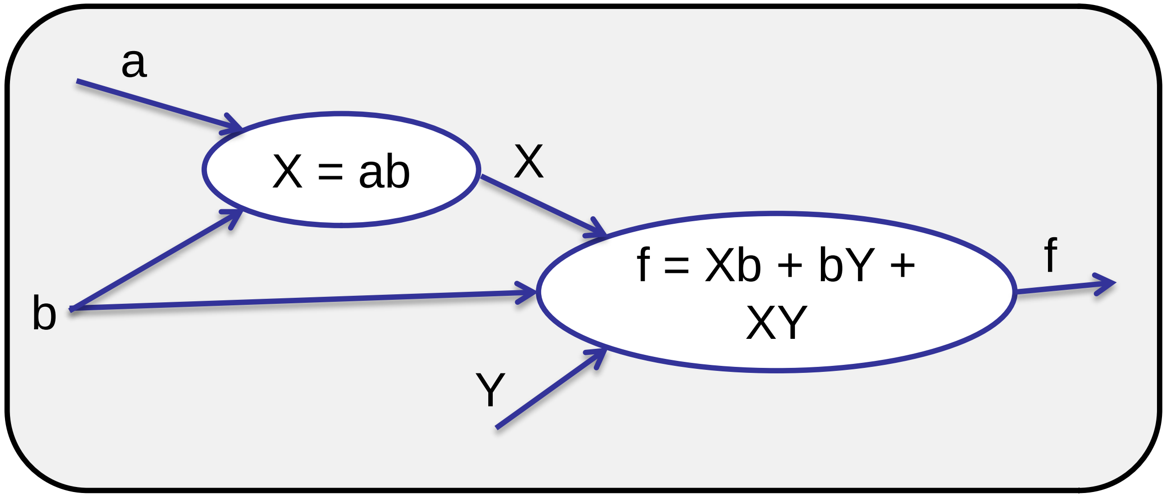
# Informal Introduction to DCs

- What if we know a little more about the network structure?
  - Now, can we say something about DCs for node f?
  - YES! Look at "impossible" values of a, b, X

| abX | possible? |
|---|---|
| 000 | |
| 001 | |
| 010 | |
| 011 | |
| 100 | |
| 101 | |
| 110 | |
| 111 | |

⇓

| XbY | possible? |
|---|---|
| 000 | |
| 001 | |
| 010 | |
| 011 | |
| 100 | |
| 101 | |
| 110 | |
| 111 | |



# Informal Introduction to DCs

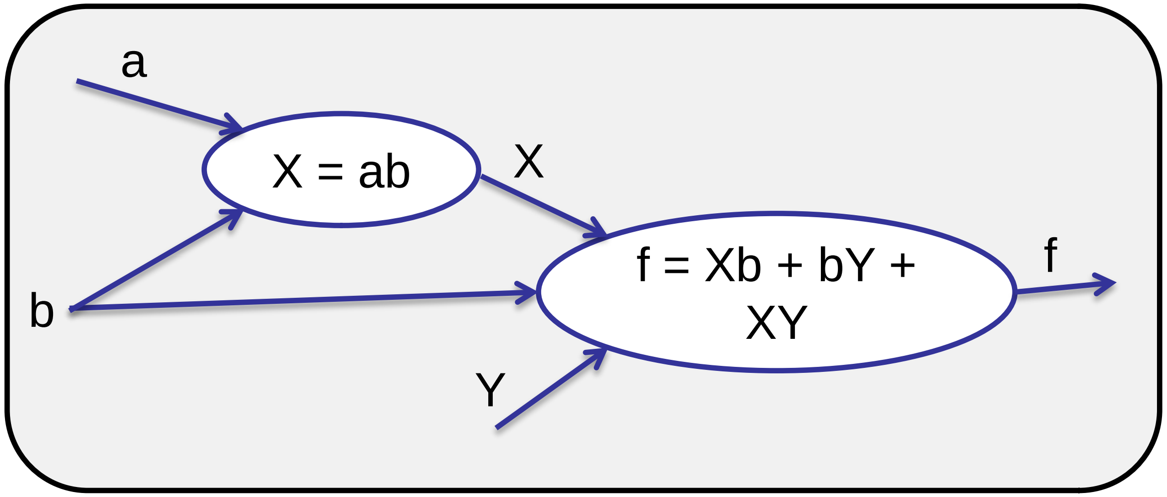
- How can we simplify f using these don't cares?

f = Xb + bY + XY

| Y \ Xb | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | | 1 | |
| 1 | | 1 | 1 | 1 |

Add don't cares & simplify

f =



## Lecture #17: Summary

- Finished the discussion on Algebraic optimization
  - Ping-pong algorithm for rectangle generation
- Boolean Optimization
  - Boolean division and factoring are much harder than algebraic counterparts
  - Number of Boolean divisors/factors can be very large
  - Boolean division using 2-level minimization
- Boolean Optimization Using Implicit Don’t Cares
  - Implicit don’t cares are introduced due to the network itself
  - We saw how to derive don’t cares at a node’s inputs by looking at it’s fanin nodes
  - A node can be simplified using its implicit don’t cares