# Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia
UC Berkeley
EECS 144/244



**Boolean Algebra and Logic Optimization- 2**

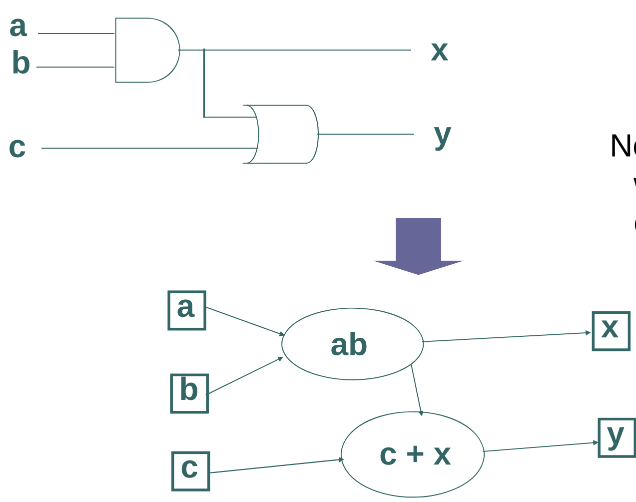
Thanks to S. Devadas, K. Keutzer, S. Malik, R. Rutenbar for several slides

## 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 #(literals), depth, …

## Tech.-Independent Multi-Level Optimization: Operations on Boolean Network

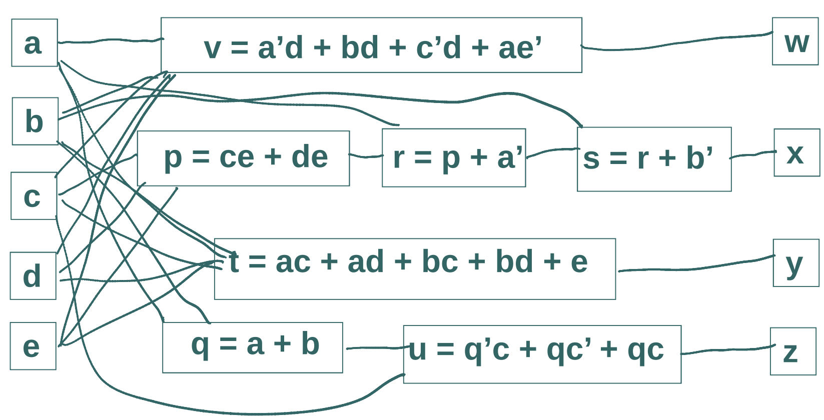
Involves performing the following operations "iteratively" until "good enough" result is obtained:

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



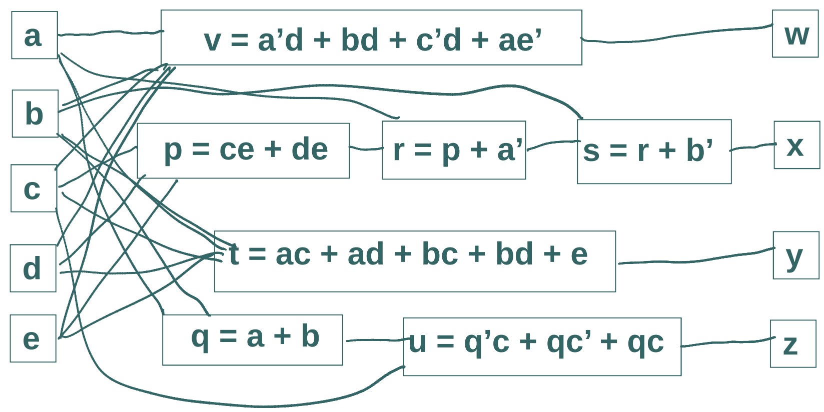
## Example

**(due to G. De Micheli)**




**#literals = 33, depth = 3**

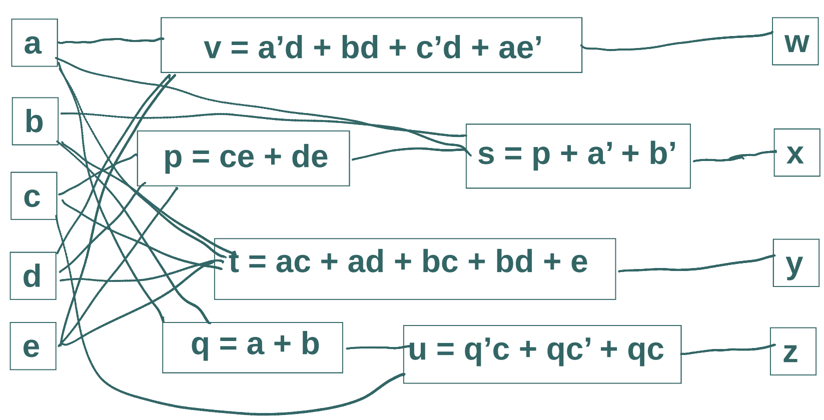
## Example: Elimination




**#literals = 33, depth = 3**

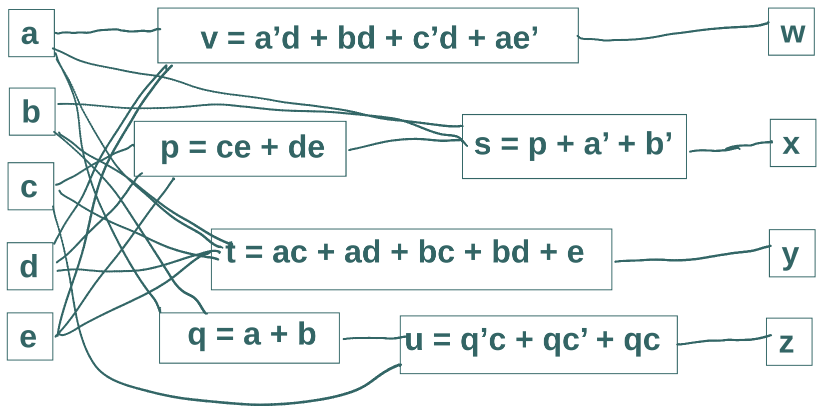


## Example: Eliminate node r




**#literals = 32, depth = 2**

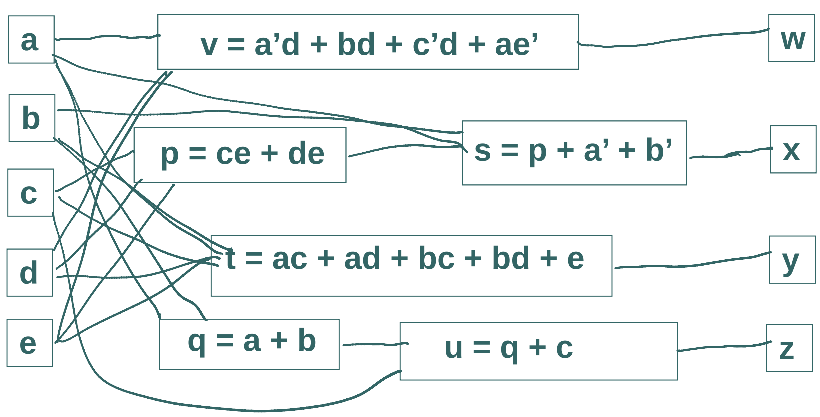
## Example: Simplification




**#literals = 32, depth = 2**

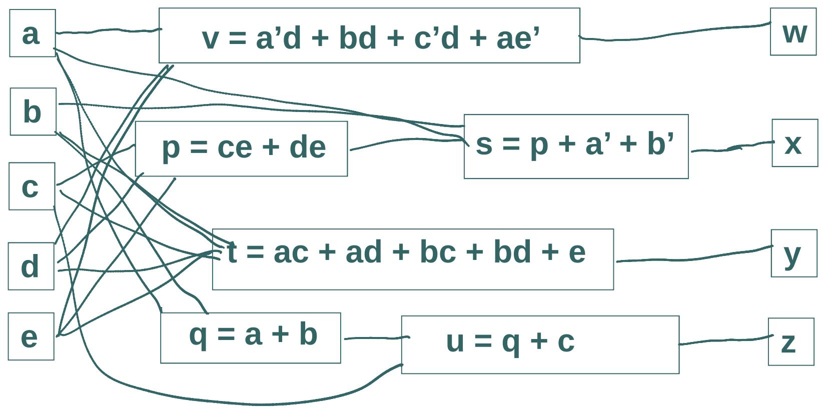


## Example: Simplifying node u




**#literals = 28, depth = 2**

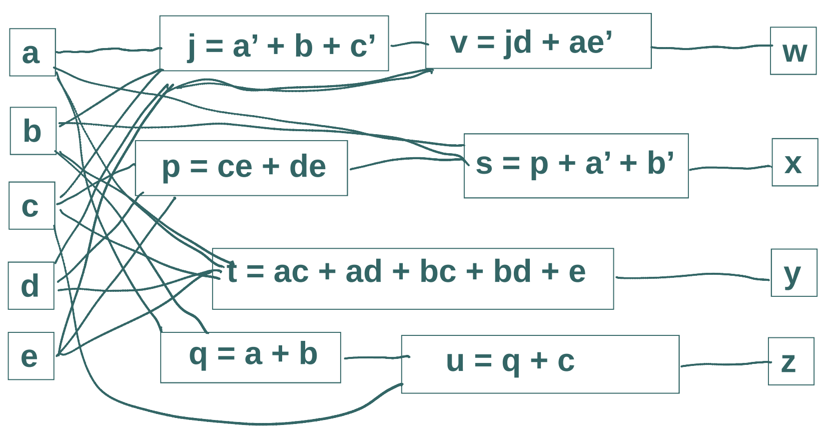
## Example: Decomposition




**#literals = 28, depth = 2**

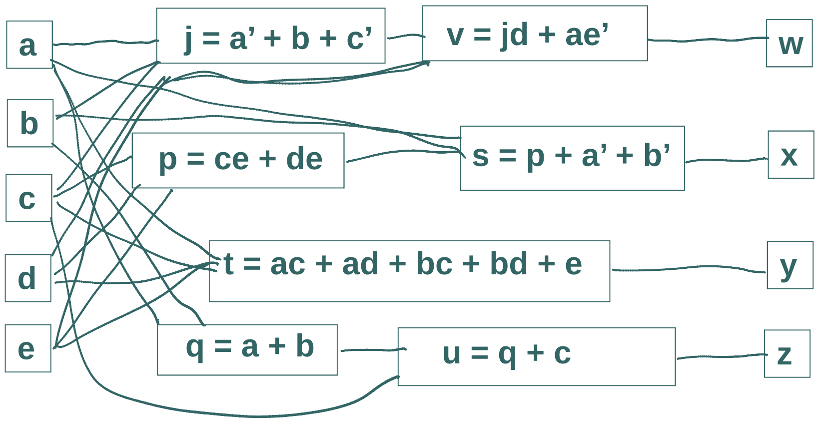


## Example: Decomposing node v




**#literals = 27, depth = 2**

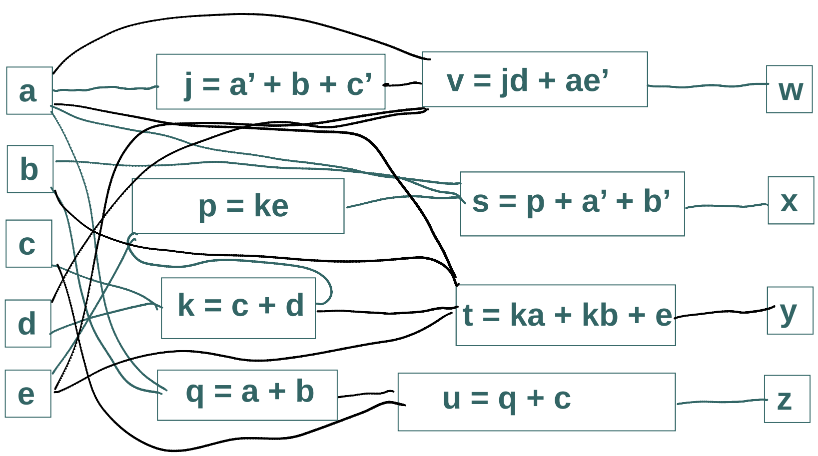
## Example: Extraction




**#literals = 27, depth = 2**

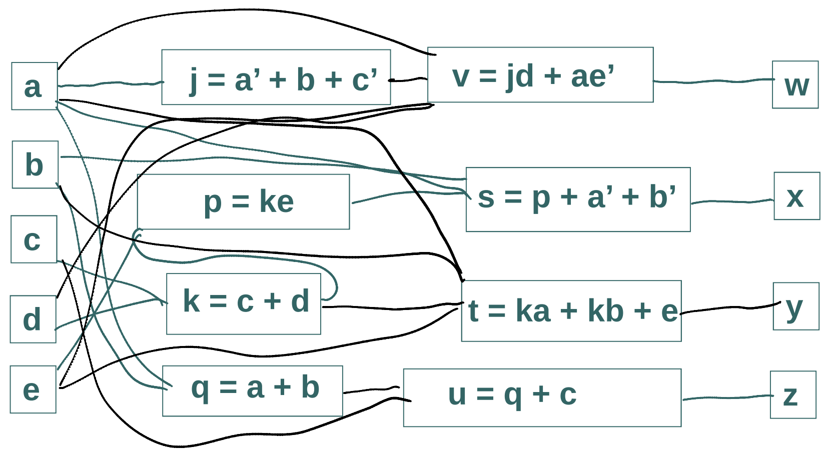


## Example: Extracting from p and t




**#literals = 23, depth = 3**

## Example: What next? Can we improve further?




**#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 . 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

$F = D . Q + R$

Definition:

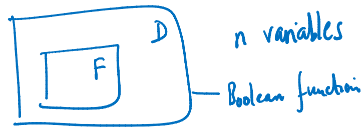
D is a Boolean divisor of F if Q and R exist such that $F = DQ + R$, $DQ \neq 0$. ($F \neq$ **0)**

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?

$2^{2^n}$






## 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 $\subseteq$ 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.b = b.a

a+b = b+a

a.(b.c) = (a.b).c

a+(b+c) = (a+b)+c

a.(b+c) = a.b + a.c

a.1 = a    a.0 = 0  a+0 = a

Boolean Algebra

a.b = b.a

a+b = b+a

a.(b.c) = (a.b).c

a+(b+c) = (a+b)+c

a.(b+c) = a.b + a.c

a.1 = a    a.0 = 0  a+0 = a

a+(b.c) = (a+b). (a+c)

a+a' = 1    a.a' = 0    a.a = a

a+a=a

a+1 = 1   a+ab = a    a.(a+b) = a

…



## 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    as    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 → a performed

## Algebraic vs. Boolean factorization

$$f = a\bar{b} + a\bar{c} + b\bar{a} + b\bar{c} + c\bar{a} + c\bar{b}$$

Algebraic factorization produces

$$f = a(\bar{b} + \bar{c}) + \bar{a}(b + c) + b\bar{c} + c\bar{b}$$

Boolean factorization produces

$$f = (a + b + c)(\bar{a} + \bar{b} + \bar{c})$$



## Algebraic Division Example

F = ac + ad + bc + bd + ae

Find Q, R, where F = D Q + R for

1. D = a + b
2. D = a

## Algebraic Division Algorithm

What we want: |F| |D|

Given F, D, find Q, R

F, D expressed as sets of cubes (same for Q, R)

Approach:

```
For each cube C in D {                      |F|
    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;             |F|
    else Q = Q ∩ B;
  }
  R = F \ (Q x D);
```

|D|

|F|

**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?

- F1 = a + b ✓
- F2 = abc + abd ✗



## Kernels: Examples

F = ae + be + cde + ab

K(f)

| Kernel | Co-kernel |
|---|---|
| {a,b,cd} | e |
| {e,b} | ? |
| ? | b |
| {ae,be,cde,ab} | ? |

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 $k_g$)

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



## Theorem, Sketched Informally

F = D1 . K1 + R1
G = D2 . K2 + R2

K1 = (X + Y + …) + stuff1
K2 = (X + Y + …) + stuff2

Then,

- F = (X + Y + …) D1 + stuff3
- G = (X + Y + …) D2 + 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

K(f)

| Kernel | Co-kernel |
|---|---|
| {a,b,cd} | e |
| {e,b} | a |
| {e,a} | b |
| {ae,be,cde,ab} | 1 |

K(g)

| Kernel | Co-kernel |
|---|---|
| {a,b} | d or e |
| {d,e} | a or b |
| {d,e,c} | b |
| {ad,ae,bd,be,bc} | 1 |



## 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 = x K + R and K = y M + S ➔ F = xy M + …
   - 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**

F/a F/b F/c F/d

**bc + bd** **ac + ad + cd** **ab + bd** **ab + bc**

q/a q/c q/d



## Finding Kernels: Example

**F = abc + abd + bcd**

F/a F/b F/c F/d

**bc + bd** **abc + ad + cd** **abc & bcd both contain c. Intersection is bc. Recurse on F/bc = a+d** **ab + bc**

**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 ∪ FindKernels(F/c);  //recursion
     }
  }
  K = K ∪ F ;
  return K;
}
```



## Reading

R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, 'Multilevel Logic Synthesis', Proceedings of the IEEE, Feb'90.