Multi-Level Logic Optimization

Multi-Level Logic Synthesis

- Two Level Logic :

$$F1 = \overline{x}yz\overline{w} + \overline{x}y\overline{z}\overline{w} + xyzw + xy\overline{z}$$
$$= \overline{x}y\overline{w} + xyzw + xy\overline{z}$$

  - Espresso
  - Programmable Logic Array (PLA)

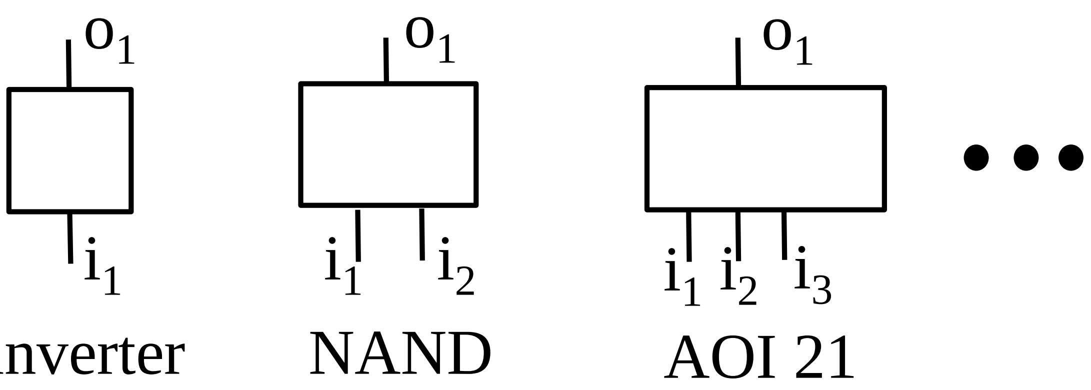
- Multilevel Logic :

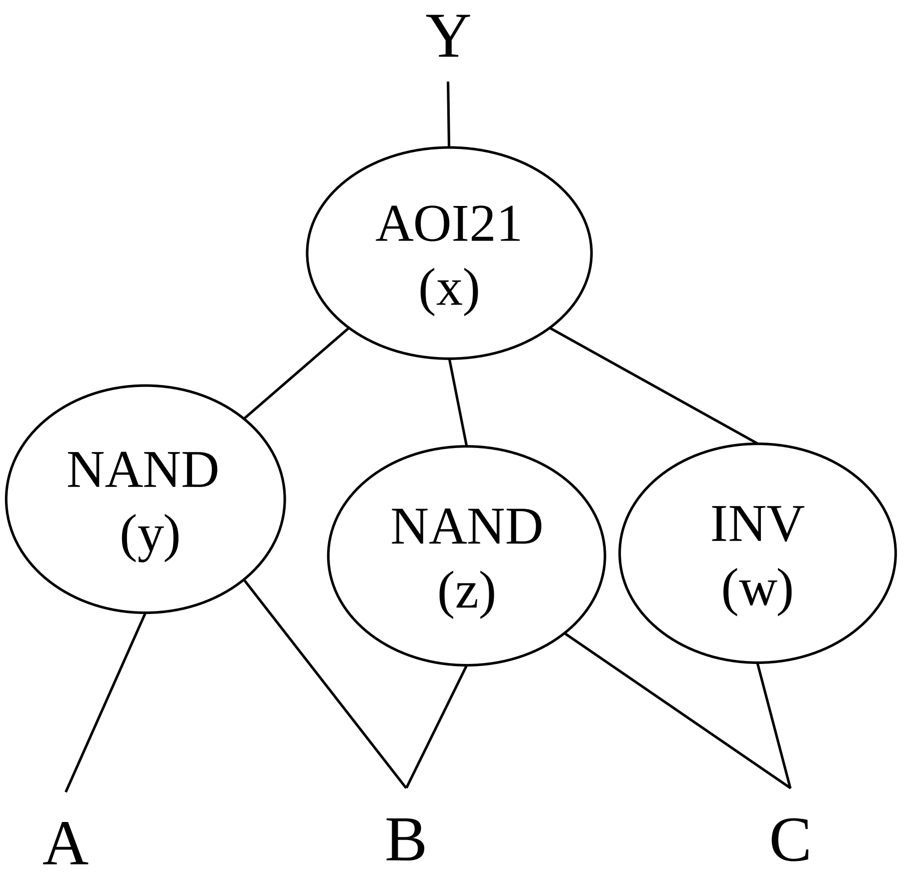
$$F2 = g(a + b\overline{c}) + c$$

  - Standard Cell

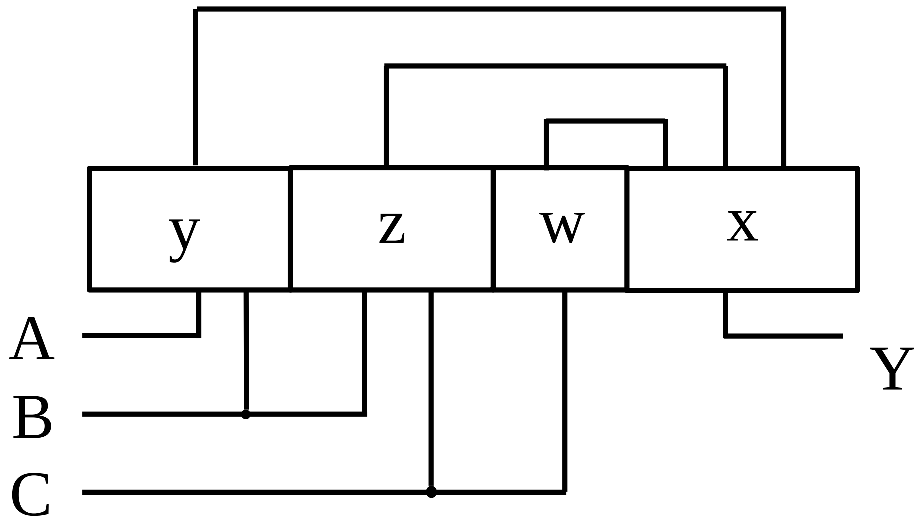
Multi-level Logic

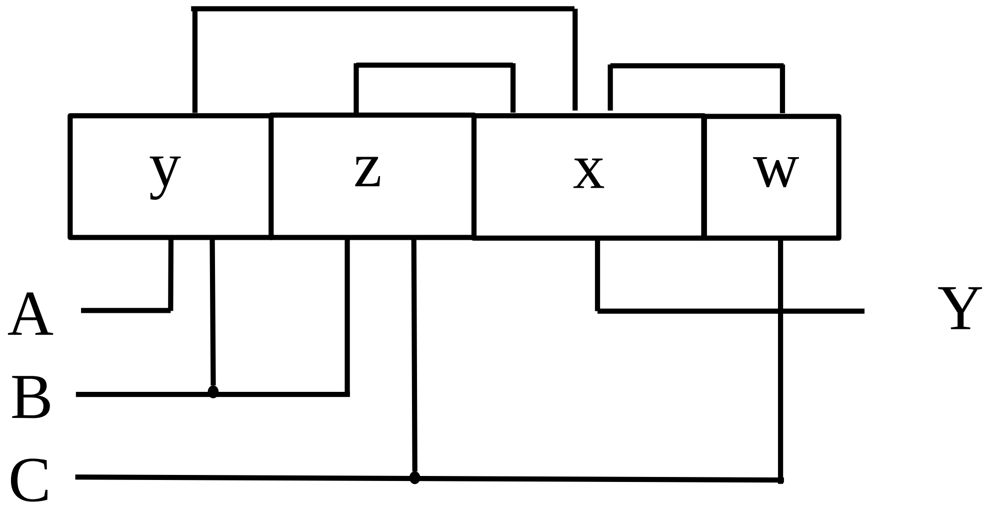
- Standard cell implementation:



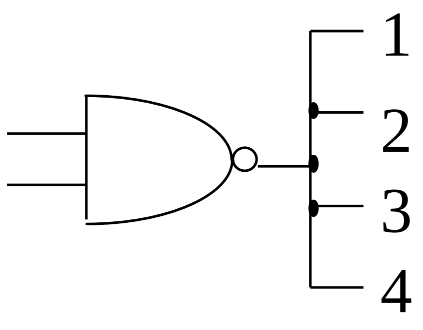


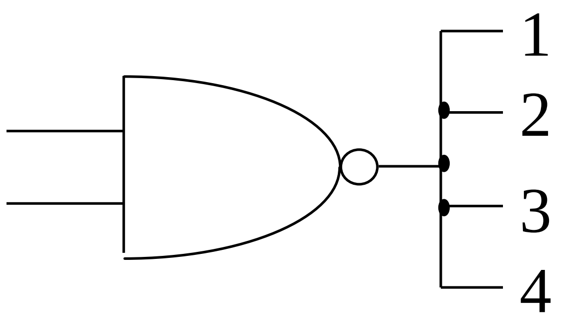
Different Placement

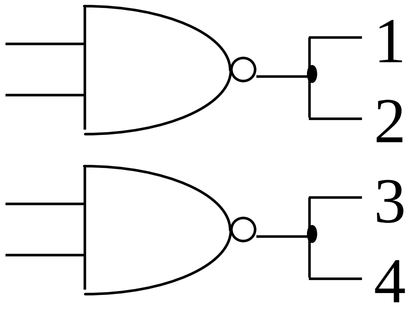


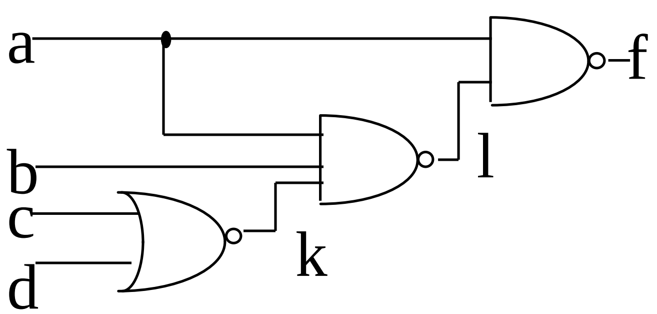


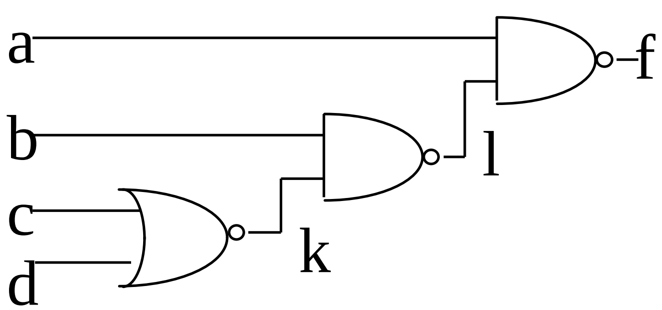
Local Optimization



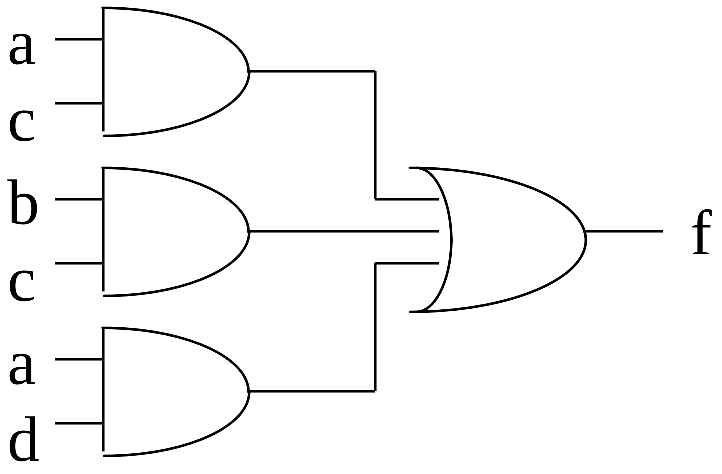


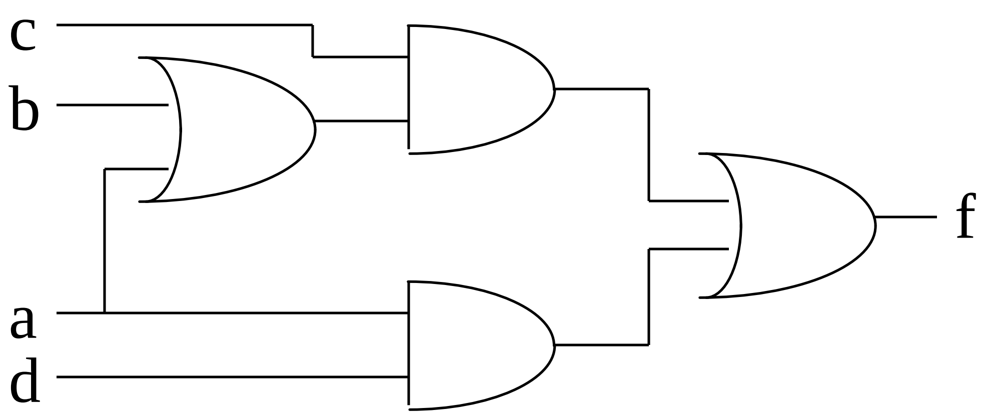






Circuit Restructuring





Representation Choices

- How to represent the function?
- How to represent the implementation?

Two-level logic :

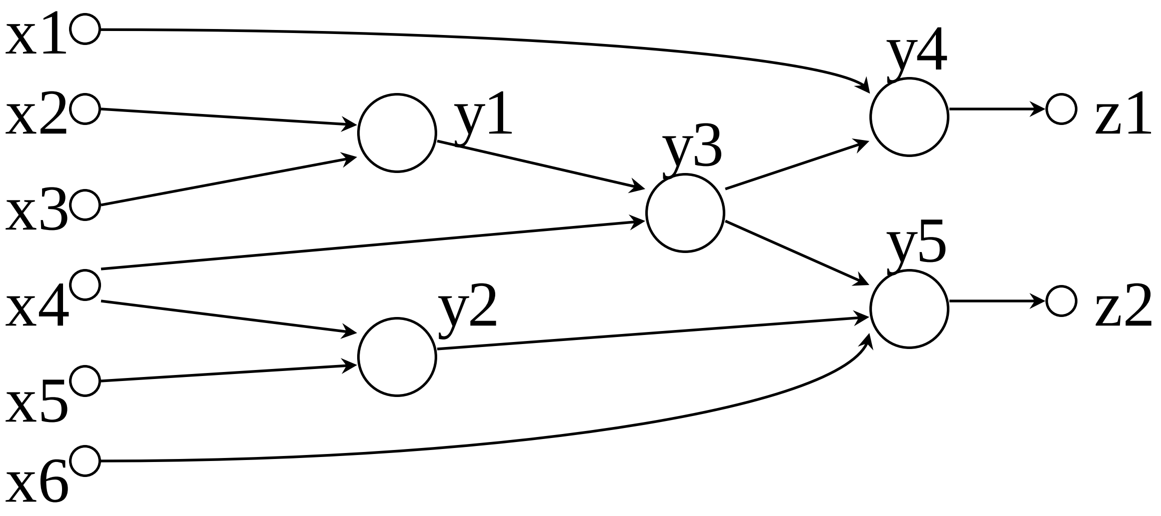
two issues are merged

Multi-level logic :

. merged view (representation and implementation are one)

. separated view

Boolean Network



$$y_1 = f_1(x_2, x_3) \qquad = x_2' + x_3'$$
$$y_2 = f_2(x_4, x_5) \qquad = x_4' + x_5'$$
$$y_3 = f_3(x_4, y_1) \qquad = x_4' y_1'$$
$$y_4 = f_4(x_1, y_3) \qquad = x_1 + y_3'$$
$$y_5 = f_5(x_6, y_2, y_3) = x_6 y_2 + x_6' y_3'$$

- Directed Acyclic Graph (DAG)
- Primary Input (PI)
- Primary Output (PO)
- Intermediate node : logic function f_i, variable y_i
- Edge
- Fan-in, transitive fan-in
- Fan-out, transitive fan-out

Node Representation

(1) Sum-of-Product

$abc'+a'bd+b'd'+b'e'f$

adv :

- easy to manipulate and minimize
- many algorithms available

disadv:

- not representative of logic complexity

  $f = ad+ae+bd+be+cd+ce$

  $f' = a'b'c'+d'e'$
- not easy to estimate if logic becoming simpler

Node Representation

(2) factored form : Any depth of sum of product

Ex: a

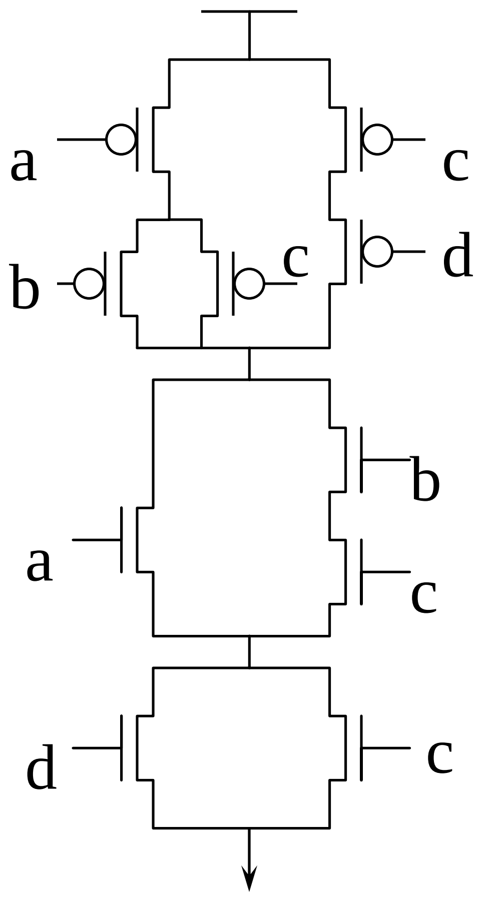
a’

ab’c

ab+c’d

(a+b)(c+a’+de)+f

Node Representation



A CMOS complex gate implementing
$f = ((a+bc)(c+d))'$
2 * literal count = transistors#

adv:

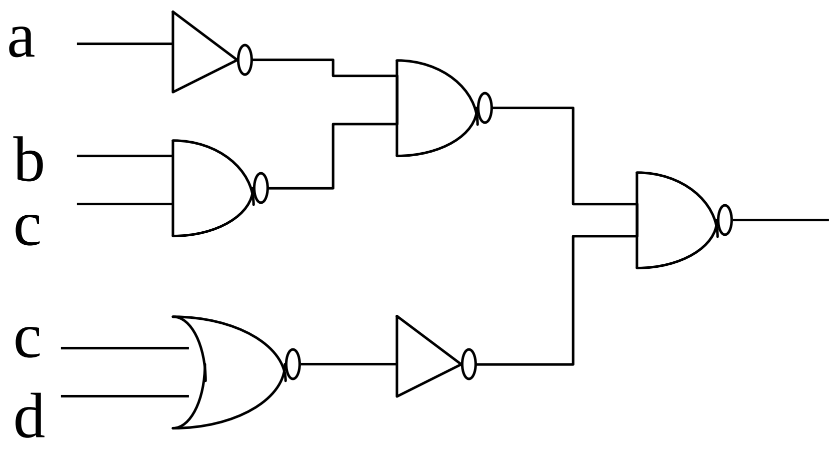
- natural multi-level representation
- good estimate of the complexity of function
- represent both the function and its complement

disadv:

- more difficult to manipulate than two-level form
- lack of the notion of optimality

Node Representation

(3) NAND or NOR form



A simple gate implementation of

$$f = ((a+bc)(c+d))'$$

adv :

- simple data structure
  storage -save
  fast simulation
- efficient optimization strategies for rule-based logic optimization
- complete with inverter count

disadv:

- The network is finely decomposed in a particular way and this may obscure some natural structures.

Multi-level Logic Optimization

- Technology independent
  - Decomposition/Restructuring

    Algebraic (Boolean) (SIS)

    Functional
  - Node optimization
- Technology dependent
  - Technology mapping

Technology Independent Phase

- Restructuring

  Basic Operations:

  1. decomposition (single function)

     f = abc+abd+a'c'd'+b'c'd'

     $\Downarrow$

     f = xy + x'y'

     x = ab

     y = c+d

  2. extraction (multiple funciton)

     f = (az+bz')cd+e

     g = (az+bz')e'

     h = cde

     $\Downarrow$

     f = xy + e

     g = xe' h = ye

     x = az+bz' y = cd

Technology Independent Phase

3. factoring (series-parallel decomposition)

$$f = ac+ad+bc+bd+e$$
$$\Downarrow$$
$$f = (a+b)(c+d)+e$$

4. substitution — boolean

$g = a+b$ — $x + x' = 1$

$f = a+bc$ — $x \cdot x' = 0$

$(a+b)(a+c)$

$f = g(a+c)$ — $=a+ac+ba+cb$

5. collapsing

$$f = ga+g'b$$
$$g = c+d$$
$$\Downarrow$$
$$f = ac+ad+bc'd'$$
$$g = c + d$$

“Division” plays a key role in all these operations.

Division

Division plays a key role in all restructuring operations

- Boolean divide
- Algebraic divide

Algebraic and Boolean Operations

- Algebraic operations:
  - Algebra of expression involving real numbers
  - Those rules that are common to the algebra of real numbers and Boolean Algebra
- Boolean operations:
  - All laws of Boolean Algebra

Boolean Algebra

- Rules hold for Boolean Algebra only
  - Idempotency

    $a \bullet a = a^2$ (real number)

    $a \bullet a = a$ (Boolean Algebra)
  - Complementation

    No direct correspondence in real field
  - Distributivity of “+” over “ • “

    $a + bc = (a+b)(a+c)$ in Boolean Algebra

    Not in real number
  - Absorption

    $a + ab = a$ in Boolean Algebra

    Not in real number

Boolean Divide

- Def 1: p is a Boolean divisor of f if $q \neq \phi$ and r exists such that

$$f = pq + r$$

(p is said to be a factor of f if $r = \phi$)

(1) q is called the quotient , f/p

(2) r is called the remainder

(3) q and r are *not unique*

Let $\mathcal{F}$ = (f, d, r)

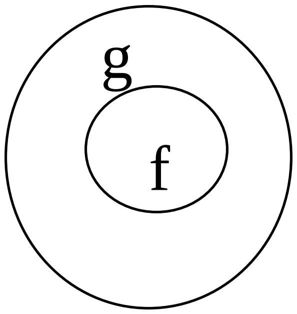
- Def 2 : g is a Boolean divisor of f if there exists h such that

$$f \subseteq gh + e \subseteq f + d \quad \text{(d : don't care)}$$

Boolean Divide

- Theorem 1

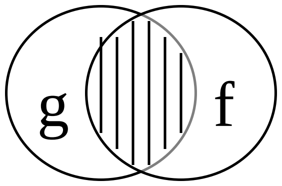
  A logic function g is a Boolean factor of a logic function of f <=> $f \subseteq g$



$f = gh$

- Theorem 2

  If $f \bullet g \neq \phi$, then g is a Boolean divisor of f.



$f = gq + r$

too many Divisor (factors)!

Algebraic Divide

- Def 3:

  f is an algebraic expression if f is a set of cubes such that no one cube contains another.

  Ex: a + ab is not an algebraic expression because a contains ab.

  ab + bd is an algebraic expression

- Def 4:

  f•g is an algebraic product if f and g are algebraic expression and have disjoint support (no input variable in common). Otherwise, f•g is a Boolean product.

  Ex:

  (a+b)(c+d) = ac+ad+bc+bd => Algebraic product

  (a+b)(a+c) = aa+ac+ba+bc => Boolean product

Weak Division

Given f and p , return q and r such that pq is an algebraic product and

$$f = pq + r$$

Algebraic Divide

- Weak-Div (f, p)

  U = set $\{U_j\}$ of cubes in f with literals not in p deleted

  V = set $\{V_j\}$ of cubes in f with literals in P deleted

  $\mathcal{U}^i = \{V_j \in V : U_j = p_i\}$

  $q = \cap \, \mathcal{U}^i$

  $r = f - pq$

Ex:

$f = ac + ad + ae + bc + bd + be + a'b$

$p = a + b$

$U = a + a + a + b + b + b + b$

$V = c + d + e + c + d + e + a'$

$\mathcal{U}^a = c + d + e$

$\mathcal{U}^b = c + d + e + a'$

$q = \mathcal{U}^a \cap \mathcal{U}^b = c + d + e \quad r = f - pq = a'b$

Division

- Substitution : knows divisor ? Yes
- Extraction : knows divisor ? No
- Factor : knows divisor ? No
- Decomposition : knows divisor ? No

Boolean Division

- Theorem

  $f1 = hx + e$ be a cover of an incompletely specified function (f, d, r). Suppose

  $x'g + xg' \subseteq d$

  where g is any function. Then, $f2 = hg + e$

  is also a cover.

- Algorithm

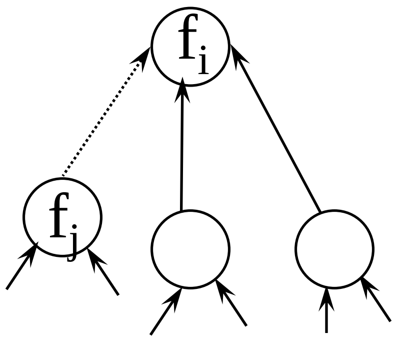
  1. $f = h \cdot g + e$ (where $h = f/g$)
  2. use a new variable x to represent g,

     $f = h \cdot x + e$
  3. form the don't care set, $xg' + x'g$
  4. minimize f with the don't care
  5. quotient f/x (quotient = the terms of f with x)

     remainder = the terms of f without x)

Substitution

- An existing node in a network may be a useful divisor in another node.



Algebraic Substitution

- Dividing the function f_i at node i by f_j or f_j' at node j pair-wise.
- If f_j is a divisor of f_i

  $f_i = g \bullet y_j + r$

No need to try all pairs
(Cases where f_j is not an algebraic divisor of f_i)

1. f_j contains a literal not in f_i
2. f_j contains more terms than f_i
3. for any literal, the count in f_j exceed that in f_i
4. f_i is f_j's transitive fan-in (cycle)

Boolean Substitution

- Ex:

  $f = a + bc$

  $g = a + b$

  substituting g into f (Let $X = a + b$)

  $DC = X(a + b)' + X'(a + b)$

  minimize $(a + bc) \bullet DC'$ (force X to appear in f)

  $\Rightarrow (a + bc)(X(a + b)' + X'(a + b))'$

  using Don't Care $= X(a + b)' + X'(a + b)$

- A minimum cover is $a + bc$. But it does not contain X or X'
- force X (or X') to remain in f

  $\Rightarrow f = a + Xc$

$f = a + gc$

$g = a + b$

Division

- Substitution : knows divisor ? Yes
- Extraction : knows divisor ? No
- Factor : knows divisor ? No
- Decomposition : knows divisor ? No

Kernel

- Kernel : for finding divisor (algebraic)
  - What is kernel?
  - Kernel algorithm
  - kernel intersection
- Too many divisor, but much smaller number of kernel.

Kernel

- Definition:

  An expression is cube-free if no cube divides the expressions evenly.

  Ex:

  a + bc is cube-free

  ab + ac is not cube-free

  abc is not cube-free

- Definition:

  The kernel of an expression f are the set of expression

  $\kappa(f) = \{ f/c \mid f/c \text{ is cube free and } c \text{ is a cube}\}$

  Ex:

  f = acb + acd + e a kernel

  f /a = cb + cd   not a kernel

  f/ac = b + d   a kernel

Kernel

- Definition :

  A cube c used to obtain the kernel k=f/c is a co-kernel c(f) denotes the set of co-kernel.

  Ex:

  f = adf + aef + bdf + bef + cdf + cef + g

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

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

Kernel

- Theorem:

  f and g have a common multiple-cube divisor d

  $\Leftrightarrow \exists \; h_f \in \mathcal{K}(f) \quad h_g \in \mathcal{K}(g)$

  such that $d = h_f \cap h_g$

  Ex:

  $f_1 = ab\,(cl + f + g) + m$

  $f_2 = ai\,(cl + f + j) + k$

  $\mathcal{K}(f_1) = \{ cl + f + g \}$

  $\mathcal{K}(f_2) = \{ cl + f + j \}$

  $\mathcal{K}(f_1) \cap \mathcal{K}(f_2) = cl + f$

  common multiple cube divisor cl + f

Kernel

- The level of a kernel

  A kernel is level-0 if it has no kernels except itself.

  A kernel is level-n if it has at least one level n-1 kernel but no kernel (except itself) of level n or higher.

  Ex:

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

kernel	level
a+b+c	0
d+e	0
(a+b+c)(d+e)	1
(a+b+c)(d+e)f + g	2

Kernel

- Why need to define level of kernel?
  - sometimes it is nearly as effective to compute a certain subset of kernel
  - computation time and quality trade off

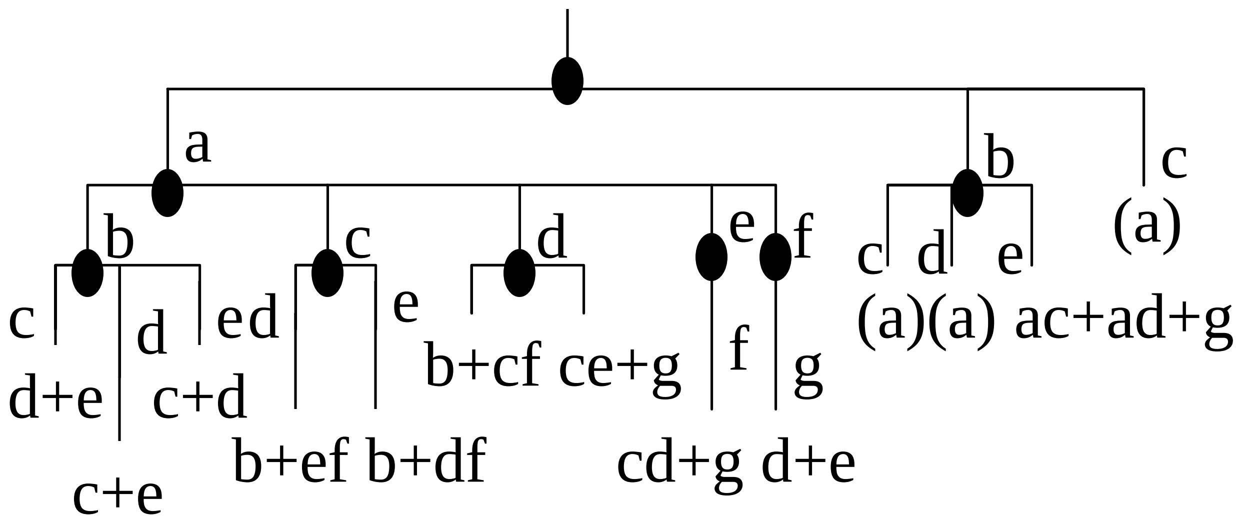
Kernel Algorithm

```
                $\downarrow$literal index     $\downarrow$expression
Kernel( j            ,               g)
    R = $\phi$
    for (i=j ; i$\leq$ n ; i++){
       if  ($l_i$ appears in more than one cube)
            c = largest cube dividing g/{$l_i$} evenly
            if  ($l_k \notin$ c for all k < i)
                R = R $\cup$ Kernel (i+1, g/({$l_i$}$\cup$ c))
     }
     R = R $\cup$ {g}
     Return R
```

- The literals in the support of f are numbered from 1 to n.

Kernel Algorithm

f = abcd+abce+adfg+aefg+abde+acdef+beg



co-kernel	kernel
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).

Kernel Intersection

- Kernel Intersection

  $K = \{ \kappa_1, \kappa_2, ... \ \kappa_n \}$
- Form a new expression IF(k) which corresponds to the set K of kernel
  - associate each distinct cube with a new literal
  - each kernel corresponds to a cube of the new function
  - Then, every element in the set of co-kernel of IF(K) corresponds to a unique kernel intersection.

Example

Ex:

Let $t_1 = abc$

$t_2 = de$

$t_3 = fg$

$t_4 = fh$

$t_5 = gh$

$K_1 = abc + de + fg = t_1t_2t_3$

$K_2 = abc + de + fh = t_1t_2t_4$

$K_3 = abc + fh + gh = t_1t_4t_5$

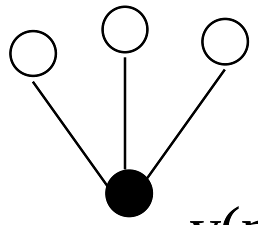
$IF(K) = t_1t_2t_3 + t_1t_2t_4 + t_1t_4t_5$

Co-kernel of (IF(k)) = $\{t_1, t_1t_2, t_1t_4\}$

Kernel Extraction

- Kernel extraction (k, n)

  1. Find all kernels of all functions and generate all kernel intersections.
  2. Choose one with best "value" .
  3. Create a new node with this as function.
  4. Algebraically substitute new node every-where.
  5. Repeat 1,2,3,4 until value $\leq$ threshold.
- Step1: Selection of level of kernel determines speed and quality trade off.
- Step2:



y(new node)

area-value(y) = freq(y) * literal(y) - literal(y) - freq(y)

Factor

- Factor(F)

```
1. If $|F| = 1$ return False
2. D = Choose_Divisor(F)
3. (Q,R) = Divide(F,D)
4. Return. Factor(Q)*Factor(D) + Factor(R)
```

- Efficiency and quality

  Step2 :

  Divisor : 1. choose literal factor
  2. choose one level-0 kernel
  3. choose the best kernel

  Step3 :

  Algebraic divide
  Boolean divide

Decomposition

- Decomposition
  - similar to factoring, except that each divisor is formed as a new node
  - For each method of factoring, we have the associated method for decomposition.

Example

Ex: $f_1 = ab(c(d+e)+f+g)+h$

$f_2 = ai(c(d+e)+f+j)+k$

– extraction(level-0 kernel)

$\alpha^0(f_1) = \{d+e\}$

$\alpha^0(f_2) = \{d+e\}$

$\alpha^0(f_1) \cap \alpha^0(f_2) = \{d+e\}$

$l = d + e$

$f_1 = ab(cl+f+g)+h$

$f_2 = ai(cl+f+j)+k$

– extraction (level-0 kernel)

$\alpha^0(f_1) = \{cl+f+g\}$

$\alpha^0(f_2) = \{cl+f+j\}$

$\alpha^0(f_1) \cap \alpha^0(f_2) = \{cl+f\}$

$m = cl+f$ $l = d+e$

$f_1 = ab(m+g)+h$

$f_2 = ai(m+j)+k$ No kernel intersection at this point

Example (cont.)

– cube extraction

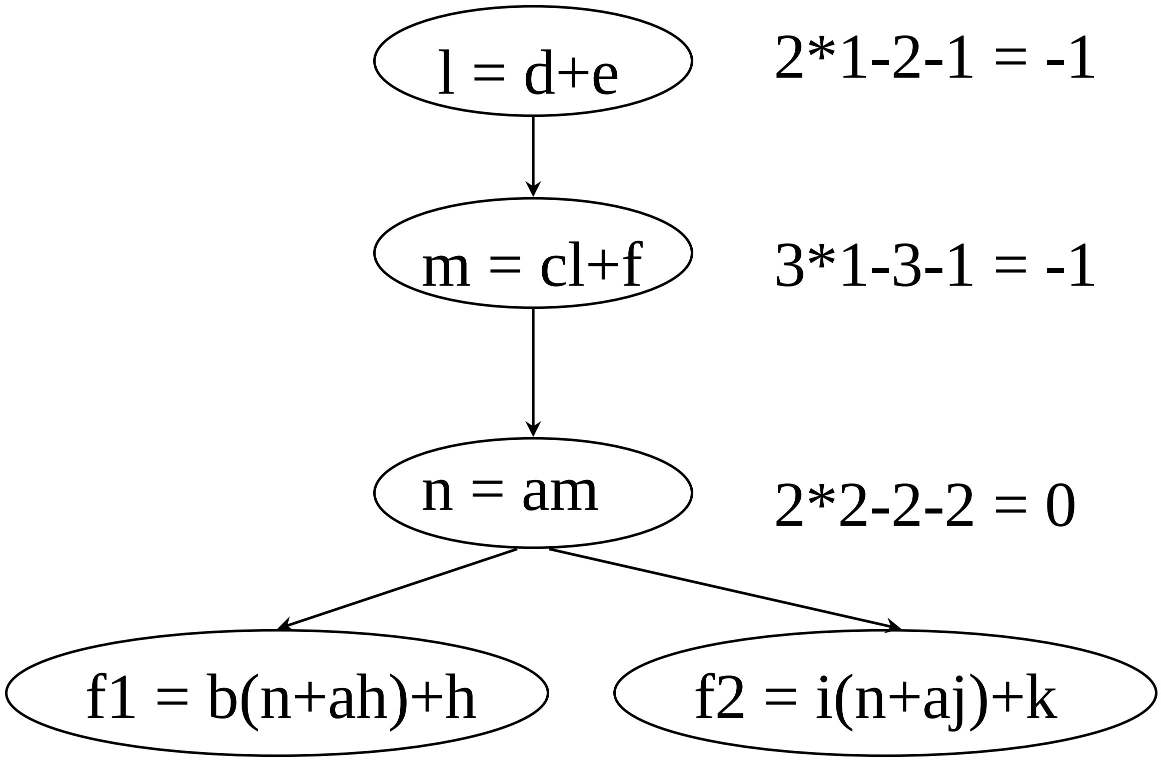
n = am

m = cl+f

l = d+e

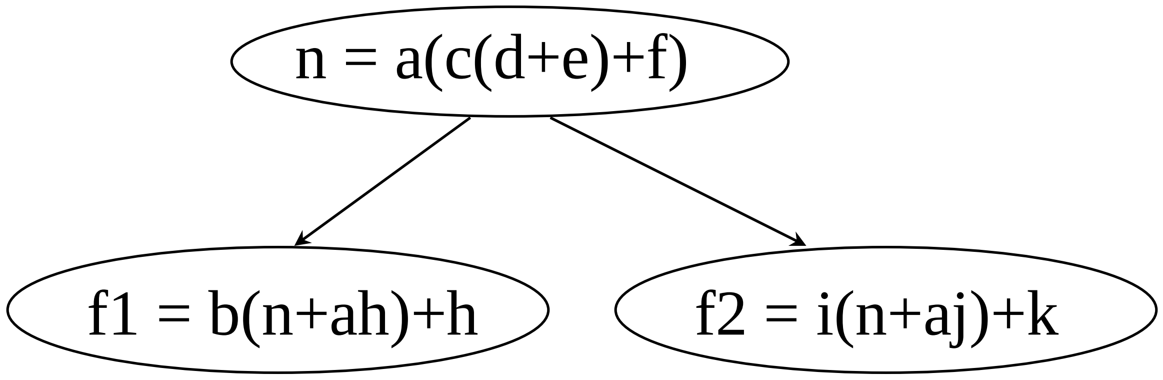
f1 = b(n+ag)+h

f2 = i(n+aj)+k



Example (cont.)

– eliminate -1 (collapsing)


n = a(c(d+e)+f)
f1 = b(n+ah)+h
f2 = i(n+aj)+k