Kungl Tekniska Högskolan

Inte
Master Pr
in System-on-Chip Design

L7: Multi-level optimization

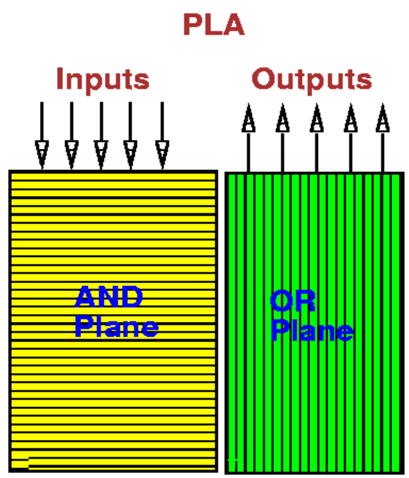
Reading material

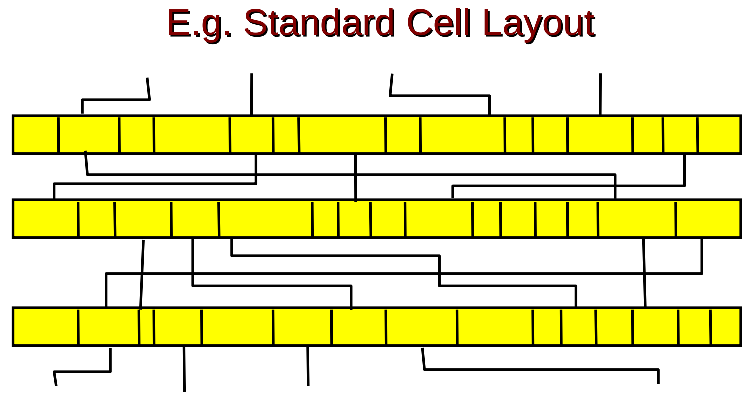
- de Micheli pp. 343 - 408
- Curtis, "The design of switching circuits", pp. 269 - 307
- Karp, "Functional decomposition and switching circuits design", J. Appl. Math, vol 11, 1963, pp. 291-335

Outline

- Introduction and Motivation
- Basic ideas in multi-level optimization
- Theory behind multi-level optimization
  – Boolean and algebraic factors
  – Kernels and kernel extraction

Two-level vs. multi-level





PLA

control logic

constrained layout

highly automatic

technology independent

Very predictable

Multi-level Logic

all logic

general (e.g. standard cell, FPGAs)

automatic

partially technology independent

Very hard to predict

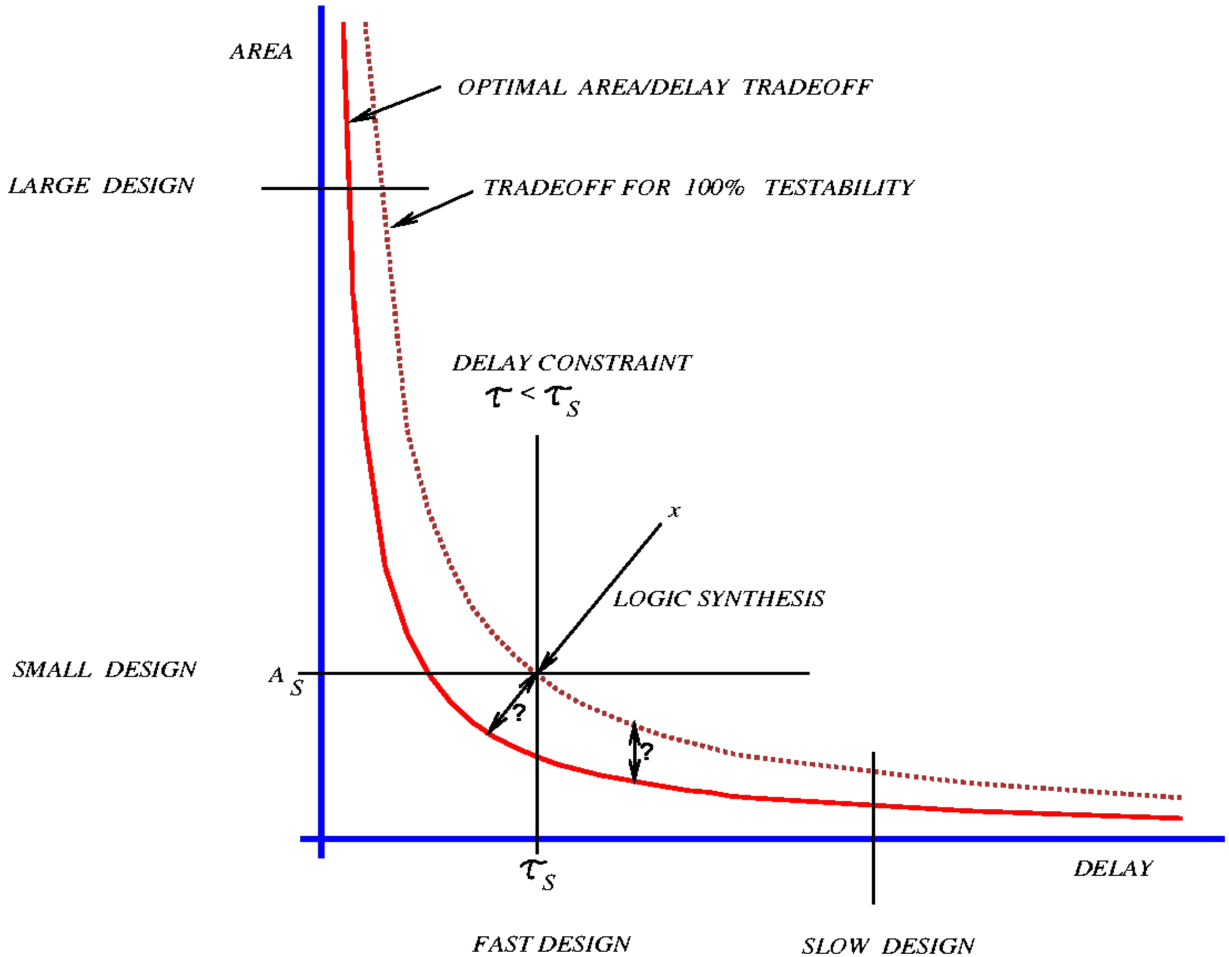
Optimization criteria for synthesis

The optimization criteria for multi-level logic is to *minimize* some function of:

- Area occupied by the logic gates and interconnect (approximated by literals = transistors in technology independent optimization)
- Critical path delay of the longest path through the logic
- Degree of testability of the circuit, measured in terms of the percentage of faults covered by a specified set of test vectors for an approximate fault model (e.g. single or multiple stuck-at faults)
- Power consumed by the logic gates
- Noise Immunity
- Place-ability, Wire-ability

while simultaneously satisfying upper or lower bound constraints placed

Example: area-delay trade-off



Network representation

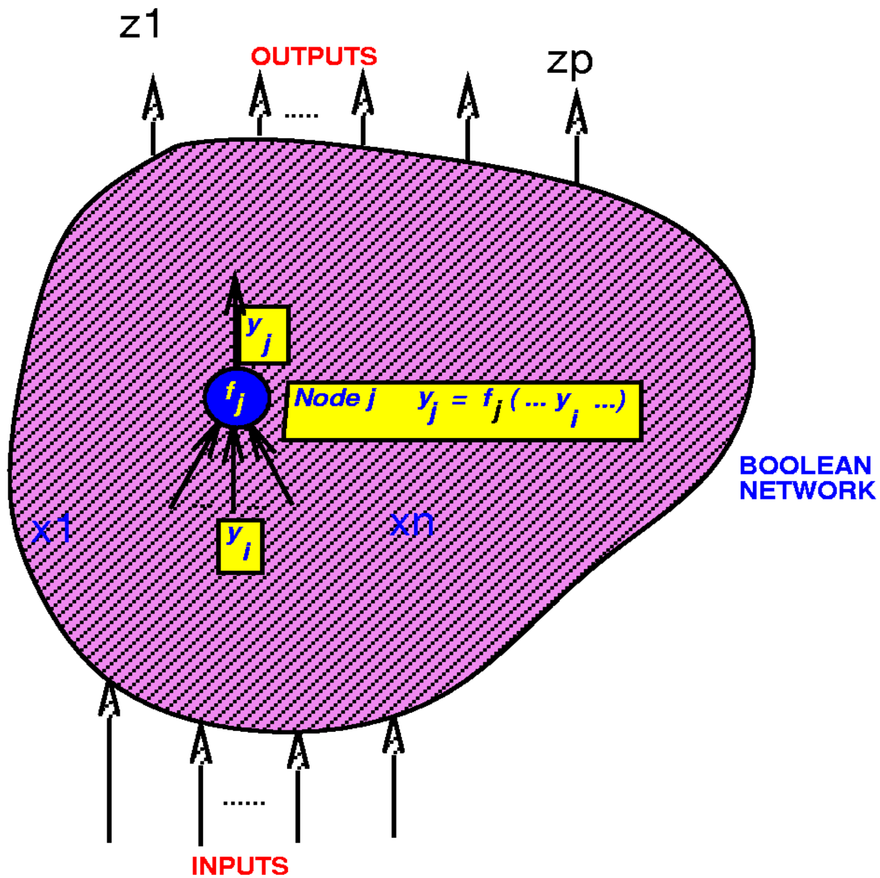
Boolean network:

- directed acyclic graph (DAG)
- node logic function representation $f_j(x,y)$
- node variable y_j: $y_j = f_j(x,y)$
- edge (i,j) if f_j depends explicitly on y_i

Inputs $x = (x_1, x_2, \ldots, x_n)$

Outputs $z = (z_1, z_2, \ldots, z_p)$

External don't cares $d_1(x), \ldots, d_p(x)$



Node representation

- Sum-of-products
- BDD
- factored forms

Sum of Products (SOP)

- Advantages:
  - easy to manipulate and minimize
  - many algorithms available
  - two-level theory applies
- Disadvantages:
  - bad representative of logic complexity.

Reduced Ordered BDDs

- given an ordering, ROBDD is canonical, hence it is a good replacement for truth tables
  - not really a good estimator for implementation complexity
- for a good ordering, BDDs remain reasonably small for complicated functions (e.g. not multipliers)
- manipulations are well defined and efficient

Factored Forms

- Advantages
  - good representative of logic complexity
  - in many designs (e.g. complex gate CMOS) the implementation of a function corresponds directly to its factored form
  - good estimator of logic implementation complexity
  - doesn't blow up easily
- Disadvantages
  - not as many algorithms available for manipulation
  - hence often just convert into SOP before manipulation

Manipulation of Boolean Networks

- Basic Techniques:
  - Global structural operations (change topology)
    - algebraic
    - Boolean
  - Local node simplification (change node functions)
    - don't cares
    - node minimization

Boolean and algebraic methods

- Boolean methods
  - exploit properties of Boolean algebra
  - use don’t cares
  - complex at times
- Algebraic methods
  - treat functions symbolically as polynomials
  - exploit properties of polynomial algebra
  - simpler and faster, but weaker

Boolean and Algebraic Methods

- In both methods, the goal is to reduce the number of literals in network representation by factorization
- “weaker” means that algebraic methods may not find the decomposition which is found by Boolean methods
- Contrary, Boolean methods will find all the decompositions found by algebraic methods

Example

- Consider the function

  $f = ab + ac + ad + a'c + a'd$

- Using algebraic method, we get:

  $f = a(b + c + d) + a'(c + d)$, 7 literals

- Using Boolean method, we get

  $f = ab + c + d$, by applying $a+a' = 1$, 4 literals

Boolean methods

- Based on the theory of Boolean decomposition
  - Ashenhurst 1959: disjoint decomposition
  - Curtis 1962: non-disjoint decomposition
  - Roth, Karp 1963 : some extensions to MVL and practical algorithms
  - von Stengel 1991: disjoint decomposition in MVL case

Problem formulation

- Given a function f, express it as a composite function of some set of new functions
- Sometimes, a composite expression can be found in which the new functions are significantly simpler than f
- The problem of selecting the "best" decomposition is too hard to be solved exhaustively

Problem formulation

- All practical algorithms using decomposition theory in logic circuit synthesis restrict the type of decomposition
- The basis for the different types of decomposition is the simple disjunctive decomposition

Simple disjunctive decomposition

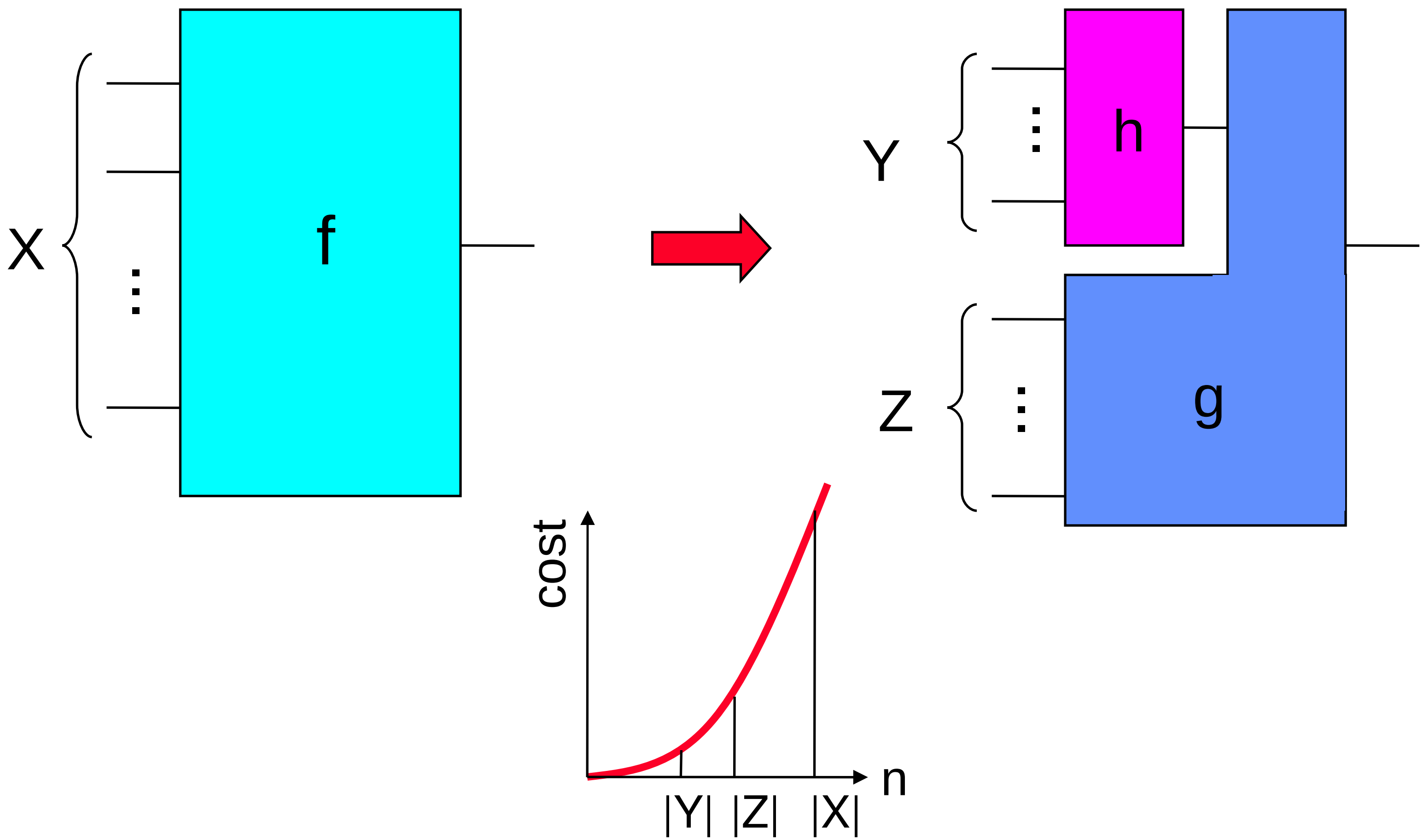
- Let $X := (x_1, \ldots, x_n)$
- Simple disjunctive decomposition of a function $f: B^n \rightarrow B$ is a representation of the form:

$$f(X) = g(h(Y), Z)$$

where $h: B^{|Y|} \rightarrow B$, $g: B^{|Z|+1} \rightarrow B$ and $Y, Z \subseteq X$ such that $Y \cup Z = X$ and $Y \cap Z = \emptyset$

- Y is called bound set; Z is called free set

Simple disjunctive decomposition



Bound set existence condition

- Suppose f = g(h(Y),Z) is given by a Karnaugh map with the columns representing the variables from Y and the rows - from Z

x_3 \ x_1x_2	00	01	11	10
0	0	1	0	0
1	1	1	1	1

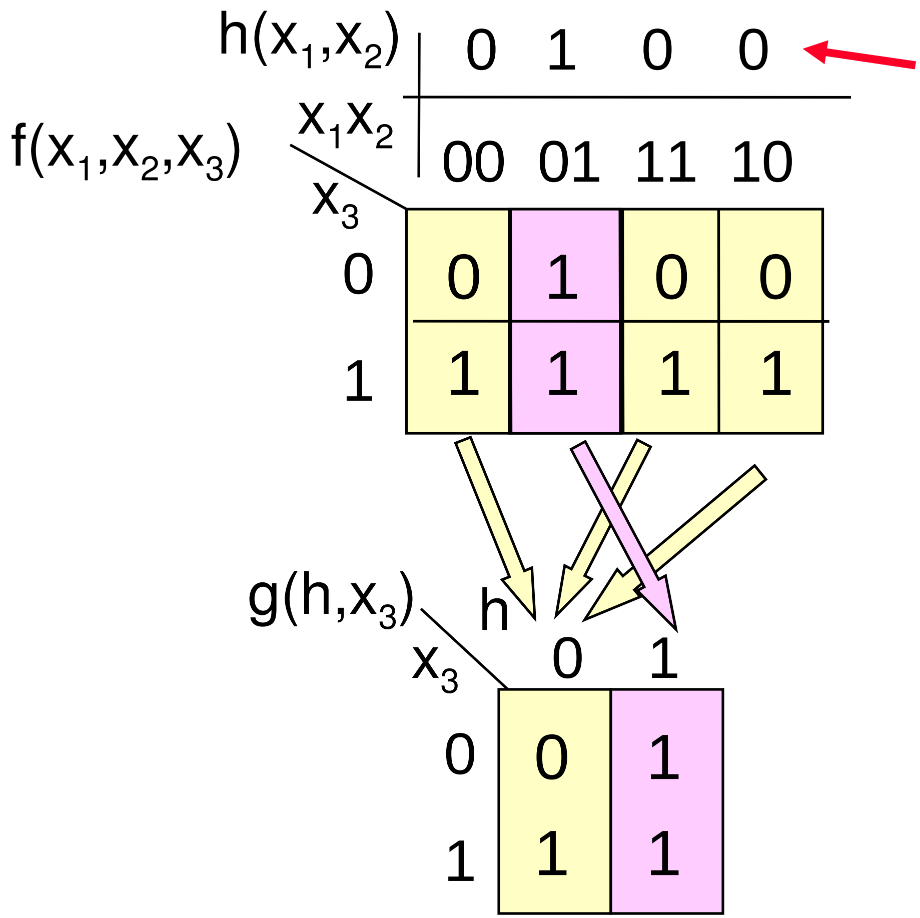
$Y = \{x_1, x_2\}$

$Z = \{x_3\}$

$k(Y/Z) = 2$

- Column multiplicity $k(Y/Z)$ is the number of distinct columns in such a map

Column multiplicity



unique up to complementation

Bound set existence condition

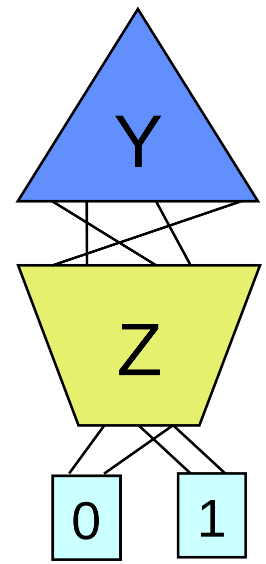
Theorem (Ashenhurst, 1959): for $f: B^n \rightarrow B$, Y is a bound set if and only if $k(Y/Z) \leq 2$

- Brute-force method for finding all bound sets:
  - build Karanugh maps for all possible partitionings Y/Z an check column multiplicity
  - N of all partitionings is $O(2^n)$ for $|X| = n$

Finding bound sets from BDDs

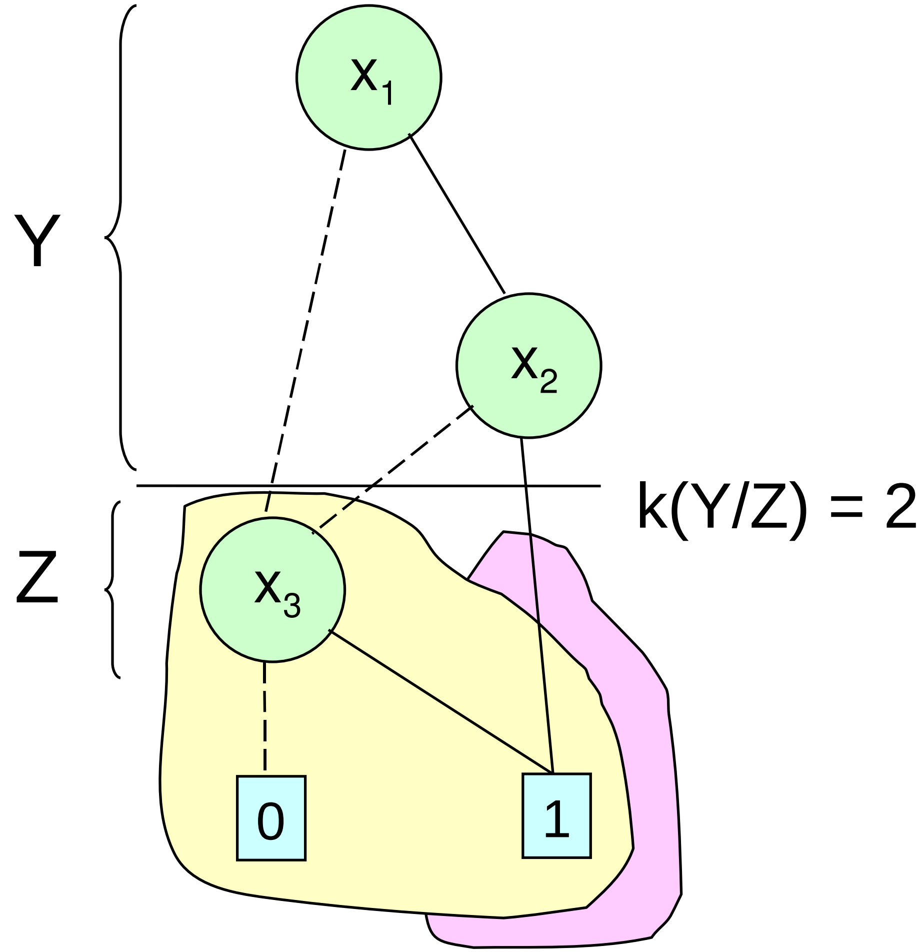
- A more efficient way to check whether Y is a bound set is to build a BDD with the variables from Y on the top:

k(Y/Z) = 4



- k(Y/Z) = = number of nodes in the lower block adjacent to the cut line

Example



x_3 \ x_1x_2	00	01	11	10
0	0	0	1	0
1	1	1	1	1

Multiple-valued functions

Theorem (Karp, 1963): for $f: M^n \rightarrow M$, Y is a bound set if and only if $k(Y/Z) \leq m$

- If we have $k(Y/Z) \leq m$ for a Boolean function, we can decompose it as:

$$f(X) = g(h(Y),Z)$$

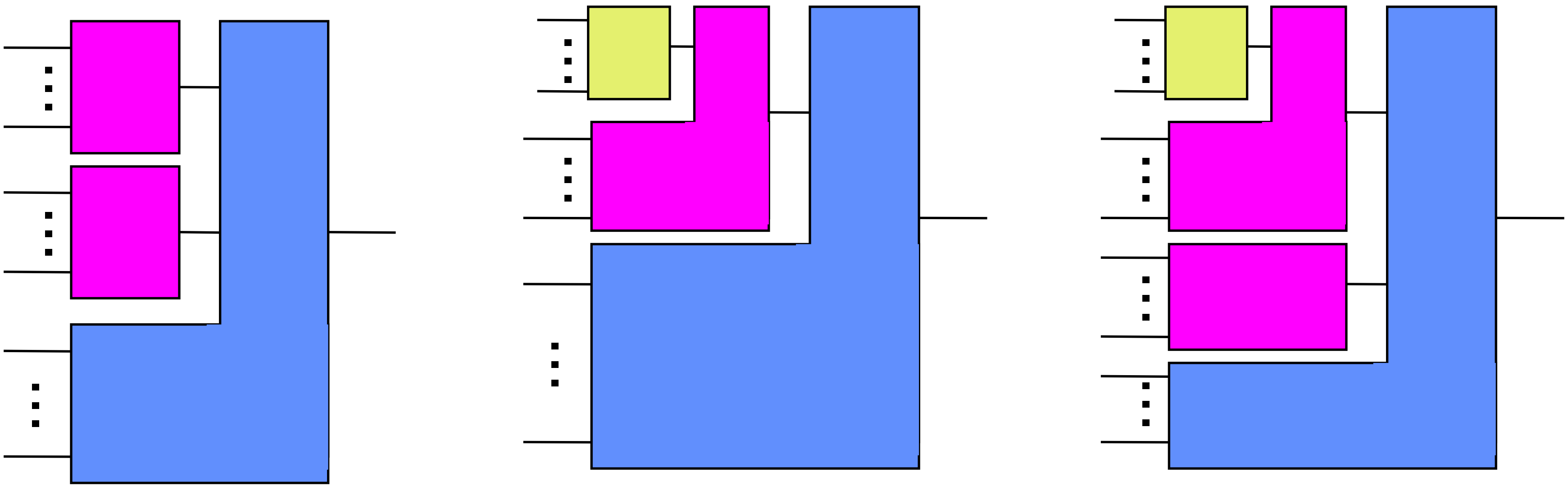
with $h: B^{|Y|} \rightarrow M$, $g: B^{|Z|} \times M \rightarrow B$, or

$$f(X) = g(h_1(Y), h_2(Y), \ldots, h_{\log_2 m}(Y), Z)$$

Complex decompositions

- Once a decomposition $f(X) = g(h(Y),Z)$ is found, either g, h, or both may be similarly decomposed, giving one of the following complex disjunctive decomposition types:
- multiple: $f(X) = g(h(Y_1), k(Y_2), Z)$
- iterative: $f(X) = g(h(k(Y_1), Z_1), Z_2)$
- tree-like : $f(X) = g(h(k(Y_1), Z_1), l(Y_2), Z_2)$

Examples of complex decompositions



multiple iterative tree-like

The “best” decomposition

- The more f is decomposed, the more its cost is reduced
- often a function can be decomposed in several different ways depending on the bound set chosen
- since a function may have up to 2^n bound sets, it is too long to consider all possible combinations
  – a theory is needed to decide which is the best

Support set

- The set of variables on which the function f actually depends is called its support set sup(f)

$$sup(f) = (x_i \mid f|_{x_i=0} \neq f|_{x_i=1})$$

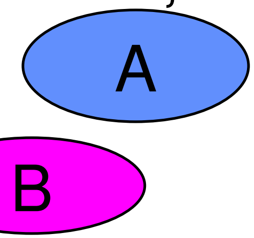
- Example: Support set of

$$f(x_1,x_2,x_3,x_4,x_5) = x_1 + x_2$$

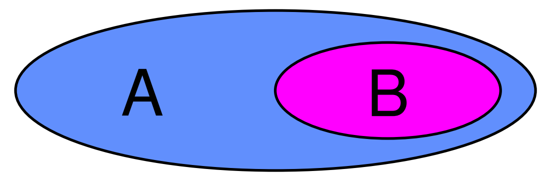
is $sup(f) = \{x_1,x_2\}$

Relation between bound sets

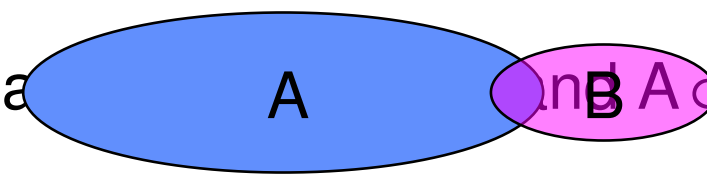
- There are 3 possible ways for two bound sets, A and B, to be related:
  - they are non-disjoint, i.e. $A \cap B = \emptyset$



  - A contains B , i.e. $A \subset B$

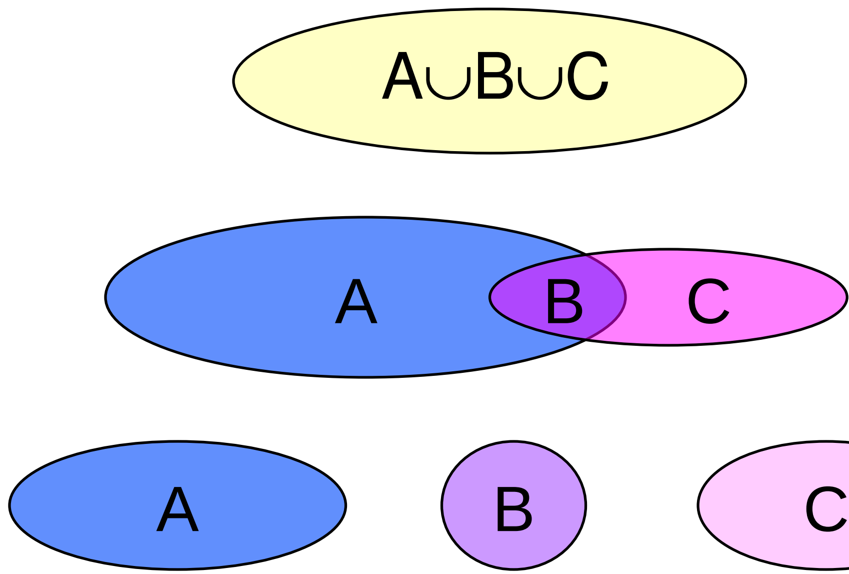


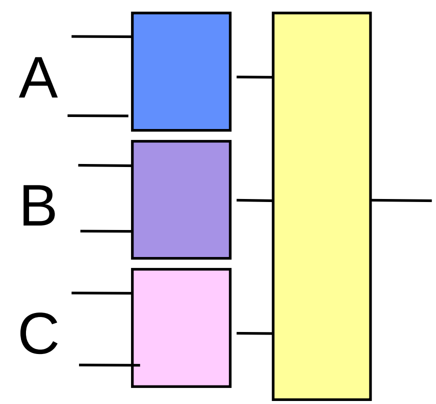
  - they overlap, i.e. $A \cap B \neq \emptyset$ and $A \not\subset B$



Fundamental Lemma

Lemma (Ashenhurst, 1959): If $A \cup B$ is a bound set and $B \cup C$ is a bound set, then A, B, C and $A \cup B \cup C$ are bound sets





Ordering

- The bound sets A and B are ordered by inclusion if and only if $A \supset B$
- Example: $\{a,b,c,d\} \supset \{b,d\} \supset \{d\}$

Composition tree

Theorem (Ashenhurst, 1959): Given a function $f: B^n \rightarrow B$ with $sup(f) = (x_1,...,x_n)$, the set of all its non-overlapping bound sets, partially ordered by inclusion, form a tree

- The tree is unique for a given function (up to complemnetation)
- The number of nodes in the tree is $O(n)$

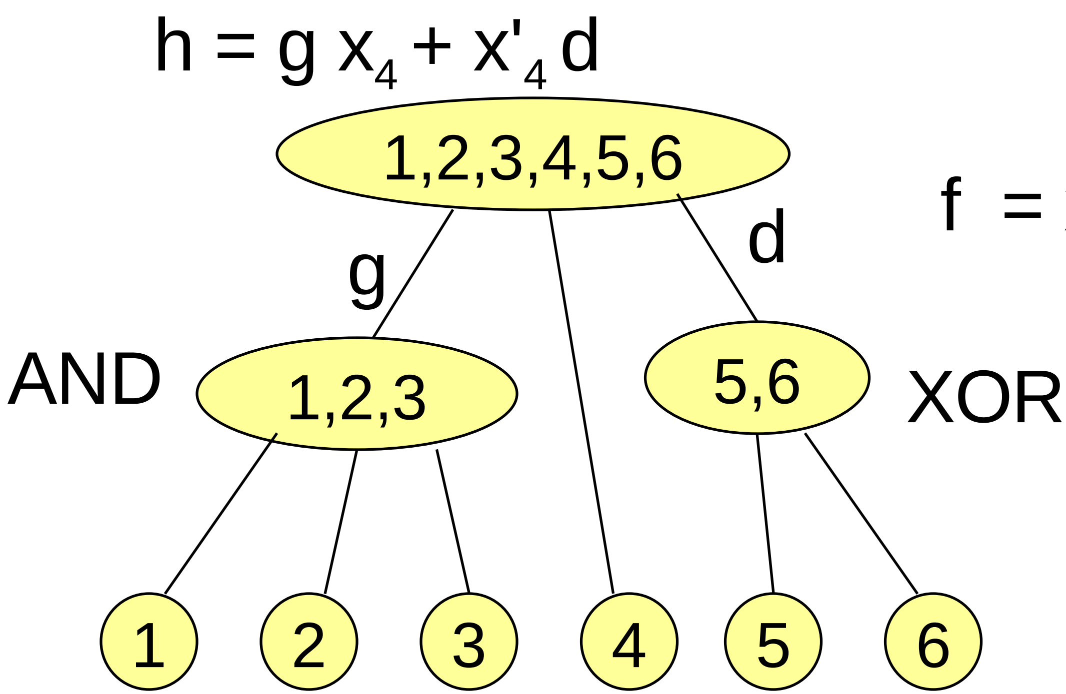
Consequences

- some bound sets can be implied

$$A \cup B \wedge B \cup C \Rightarrow A \wedge B \wedge C \wedge A \cup B \cup C$$

- if two composition trees are different, the functions they represent are not equivalent
  - checking equivalence can be terminated earlier

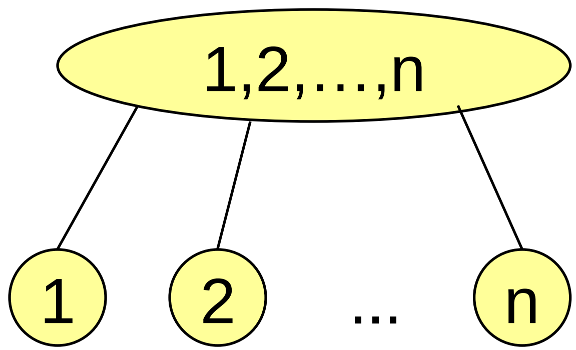
Example



$$f = x_1 x_2 x_3 x_4 + x'_4 (x_5 \oplus x_6)$$

Problem

- Some functions have trivial composition trees



Roth-Karp decomposition

Theorem (Karp, 1963): For multiple-valued functions $M^n \rightarrow M$, Y is a bound set if and only if $k(Y/Z) \leq m$

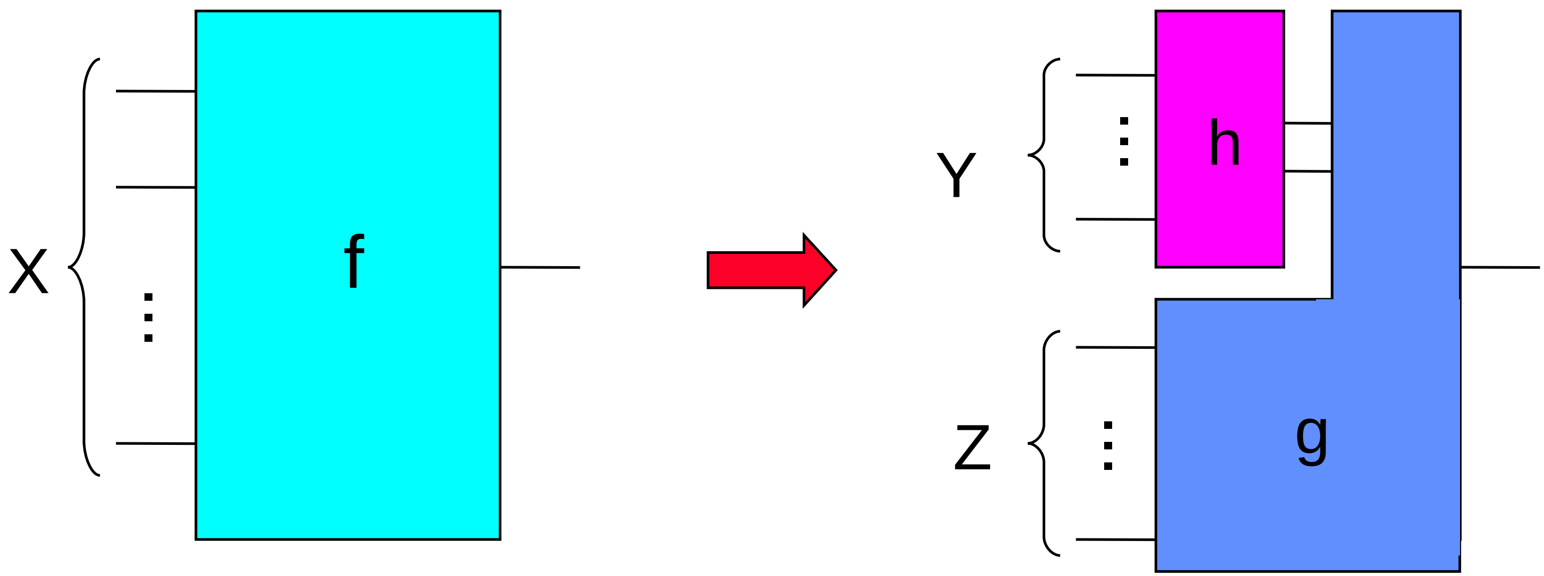
- If we have $k(Y/Z) \leq m$ for a Boolean function, we can decompose it as:

$$f(X) = g(h(Y),Z)$$

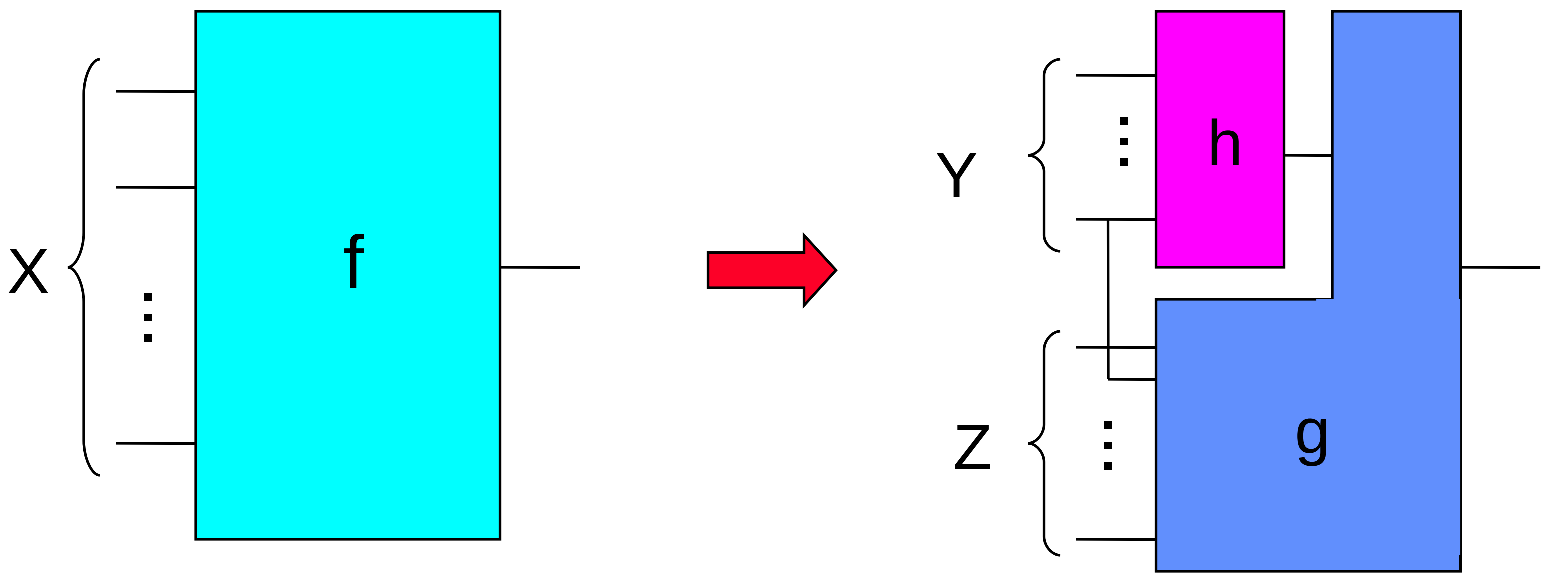
with $h: B^{|Y|} \rightarrow M$, $g: B^{|Z|} \times M \rightarrow B$, or

$$f(X) = g(h_1(Y), h_2(Y), \ldots, h_{\lceil \log_2 m \rceil}(Y), Z)$$

Example, k(Y/Z) = 4



Non-disjoint decompositions



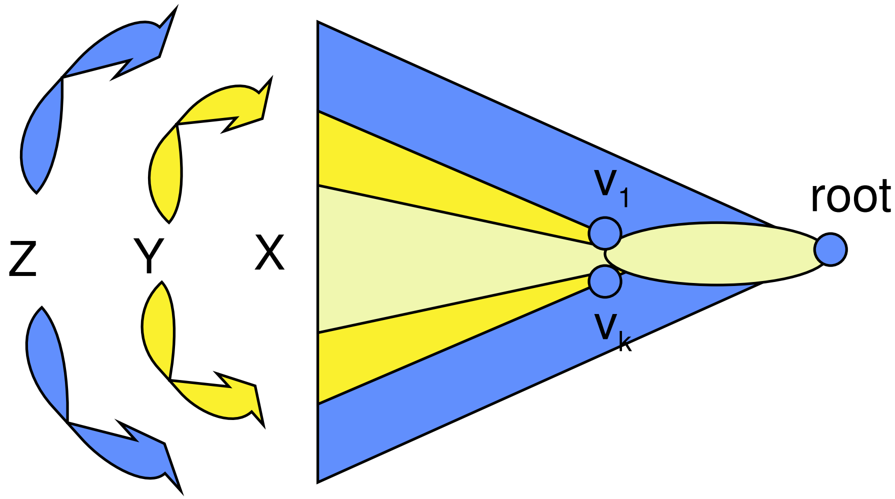
$$Y \cap Z \neq \emptyset$$

Algorithms based on Boolean decomposition

- There are algorithms for finding all bound sets and deriving from them the decomposed expression for f
  - mostly BDD based, quite fast
- For functions with no disjoint decomposition
  - Roth & Karp decomposition is used
  - Non-disjoint types of decompositions are used (harder to find)

Relation to dominators

- Let X be a set primary inputs dominated by $\{v_1,\ldots,v_k\}$
- Let $X \cup Y$ be a set primary inputs the transitive fan-in of $\{v_1,\ldots,v_k\}$



- Then, there exist a decomposition of type

$$f(X,Y,Z) = h(g_1(X,Y),\ldots, g_k(X,Y),Y,Z)$$

Algebraic decomposition

- Algebraic methods provide faster algorithms, because they treat a function like a symbolic polynomial
  - AND = multiplication, OR = addition operation, x and x' are two different variables
- There are fast methods for manipulating polynomials. The optimally is lost, but the results are quite good

Main idea

- Given a SOP, how do we generate a “good” factored form
- Division operation:
  - is central in many operations
  - find a good divisor
  - apply the actual division
    - results in quotient and remainder
- Factorization
  - factored forms have no inversion except at inputs
  - number of literals is used as size metric

Algebraic divisors and factors

- We say that $f_{divisor}$ is an algebraic divisor of $f_{divident}$ when:
  - $f_{divident} = f_{divisor} \cdot f_{quotient} + f_{reminder}$
  - $f_{divisor} \cdot f_{quotient} \neq 0$
  - $sup(f_{divisor}) \cap sup(f_{quotient}) = \varnothing$
- If $f_{reminder} = 0$, then $f_{divisor}$ is called factor of $f_{divident}$

Example

- Algebraic division:

  let $f_{divident} = ac + ad + bc + bd + e$ and $f_{divisor} = a + b$

  then $f_{quotient} = c + d$, $f_{reminder} = e$, because

  $(a+b)(c+d) + e = f_{divident}$ and $\{a,b\} \cap \{c,d\} = \varnothing$

- Boolean dvision:

  let $g_{divident} = a + bc$ and $g_{divisor} = a + b$

  $g_{divisor}$ is NOT an algebraic divisor, even though

  $g_{divident} = g_{divisor} \cdot g_{quotient}$ with $g_{quotient} = a + c$

  because $\{a,b\} \cap \{a,c\} \neq \varnothing$

Why do we need to require $sup(f_{divisor}) \cap sup(f_{quotient}) = \emptyset$

- It prevents generation of cubes that are contained in other cubes, as well as universal and void cubes
- Examples:

1) $\{a,b\} \cap \{c,d\} = \emptyset$: $(a+b)(c+d) = ac + ad + bc + bd$
2) $\{a,b\} \cap \{a,c\} \neq \emptyset$: $(a+b)(a+c) = aa + ac + ba + bc$
    - aa (universal cube) cannot be eliminated in algebraic model
3) $\{a,b\} \cap \{a,c\} \neq \emptyset$: $(a+b)(a'+c) = aa' + ac + ba' + bc$
    - aa' (void cube) cannot be eliminated in algebraic model

Generation of divisors

- The number of Boolean divisors of a function can be very large
- To find an optimal multi-level expression, we need to generate all possible divisors and choose an expression with the smallest number of literals
- Algebraic divisors are a subset of Boolean divisors, but this subset may still be large

Generation of divisors

- An important subset of algebraic divisors can be generated by treating cubes as divisors
- The quotient in this process is called kernel and the cube used for division is called co-kernel
- kernels and co-kernels can be used to write expressions in factorized form

Kernel

- Cube free expression is an expression which cannot be factored by a cube
  - single cubes are never cube-free
- A kernel of an expression is the cube free quotient of the expression obtained by dividing it with a cube
- Cube used to get the kernel of the expression is called its co-kernel
- Kernel set K(f) is the set of all kernels of f

Example

Let f_x = ace + bce + de + g

1. By dividing f_x by cube a we get ce
   – ce is not cube free (can be divided by c or e), so it is not kernel
2. By dividing f_x by e we get ac + bc + d
   – ac + bc + d is cube free (cannot be divided by any cube without reminder), so it is a kernel, and e is a co-kernel
3. $K(f_x)$={(ace+bce+de+g),(ac+bc+d),(a+b)}

Kernel set computation

- Naive method:
  - divide function by elements in power set of its support set
  - weed out non cube free quotients
- Smart way:
  - use recursion
    - kernels of kernels are kernels
  - exploit commutativity of multiplication
    - ab = ba

Example

Let f_x = ace + bce + de + g

1. Select kernel ac + bc + d
2. Decompose f_x as $f_x = f_y e + g$, with f_y= ac+bc+d
3. Recur on the quotient f_y:
   1. Select kernel a + b
   2. Decompose f_y as $f_y = f_z c+d$, with f_z=a+b
4. Resulting factorized expression for f_x:

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

Summary of algebraic methods

- Boolean function is treated symbolically as a polynomial
- fast manipulation algorithms
- some optimality is lost, because some Boolean properties are neglected
- useful to reduce large networks