LIBRARY BINDING

©*Giovanni De Micheli*

Stanford University

Outline

- Modeling and problem analysis.
- Rule-based systems for library binding.
- Algorithms for library binding:
  - *Structural covering/matching.*
  - *Boolean covering/matching.*
- Concurrent optimization and binding.

Library binding

- Given an unbound logic network and a set of library cells:
  - Transform into an interconnection of instances of library cells.
  - Optimize *area*, (under *delay* constraints.)
  - Optimize *delay*, (under *area* constraints.)
  - Optimize *power*, (under *delay* constraints.)
- Called also *technology mapping*:
  - Method used for re-designing circuits in different technologies.

Library models

- Combinational elements:
  - Single-output functions:
    * e.g. AND, OR, AOI.
  - Compound cells: e.g. adders, encoders.
- Sequential elements:
  - Registers, counters.
- Miscellaneous:
  - Schmitt triggers.

Major approaches

- Rule-based systems:
  - Mimic designer activity.
  - Handle all types of cells.
- Heuristic algorithms:
  - Restricted to single-output combinational cells.
- Most tools use a combination of both.

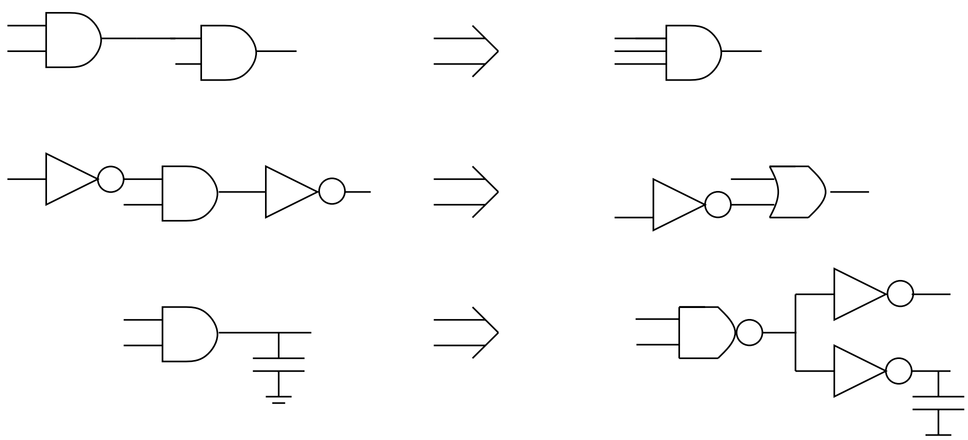
Rule-based library binding

- Binding by stepwise transformations.

- Data-base:
  - Set of patterns associated with best implementation.

- Rules:
  - Select subnetwork to be mapped.
  - Handle high-fanout problems, buffering, etc.

Example



Strategies

- Search for a sequence of transformations.

- Search space:

  - *Breadth* (options at each step).

  - *Depth* (look-ahead).

- *Meta-rules* determine dynamically breadth and depth.

Rule-based library binding

- Advantages:
  - Applicable to all kinds of libraries.

- Disadvantages:
  - Large rule data-base:
    - Completeness issue.
    - Formal properties of bound network.
  - Data-base updates.

Algorithms for library binding

- Mainly for single-output combinational cells.
- Fast and efficient:
  - Quality comparable to rule-based systems.
- Library description/update is simple:
  - Each cell modeled by its function or equivalent pattern.

Problem analysis

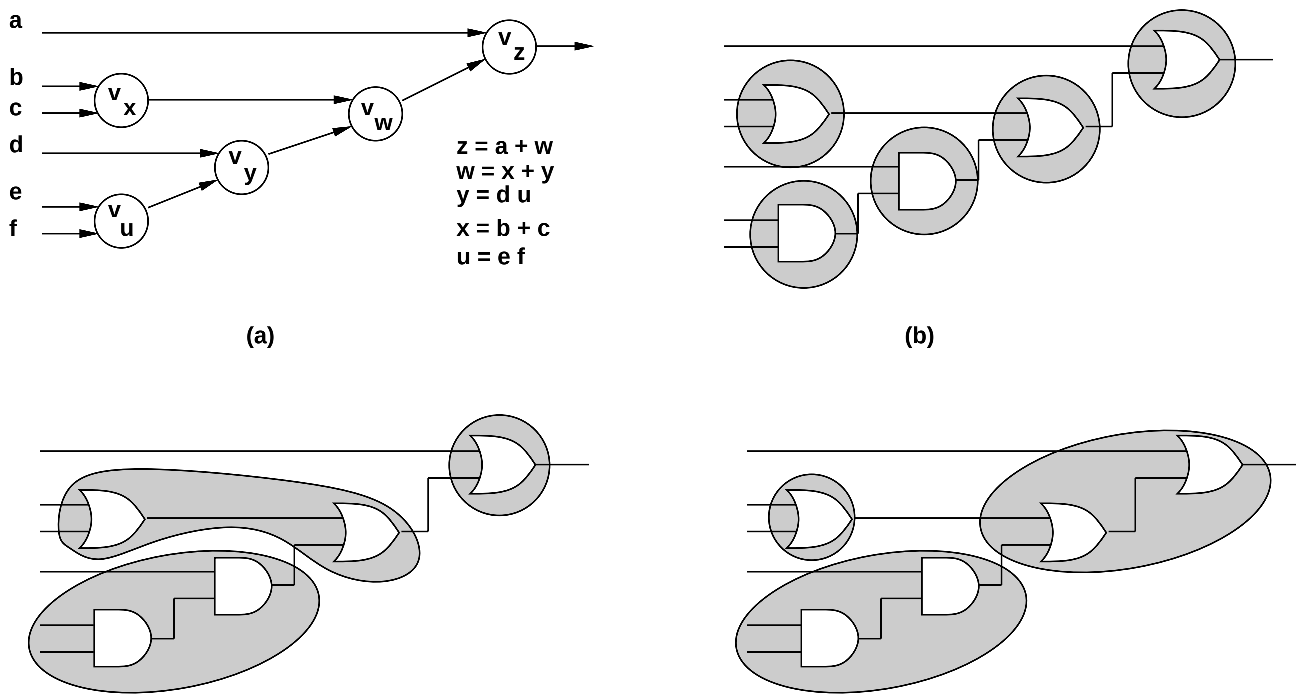
- Matching:
  - A cell matches a sub-network
    if their terminal behavior is the same.
  - Input-variable *assignment* problem.

- Covering:
  - A cover of an unbound network
    is a partition into subnetworks
    which can be replaced by library cells.

Assumptions

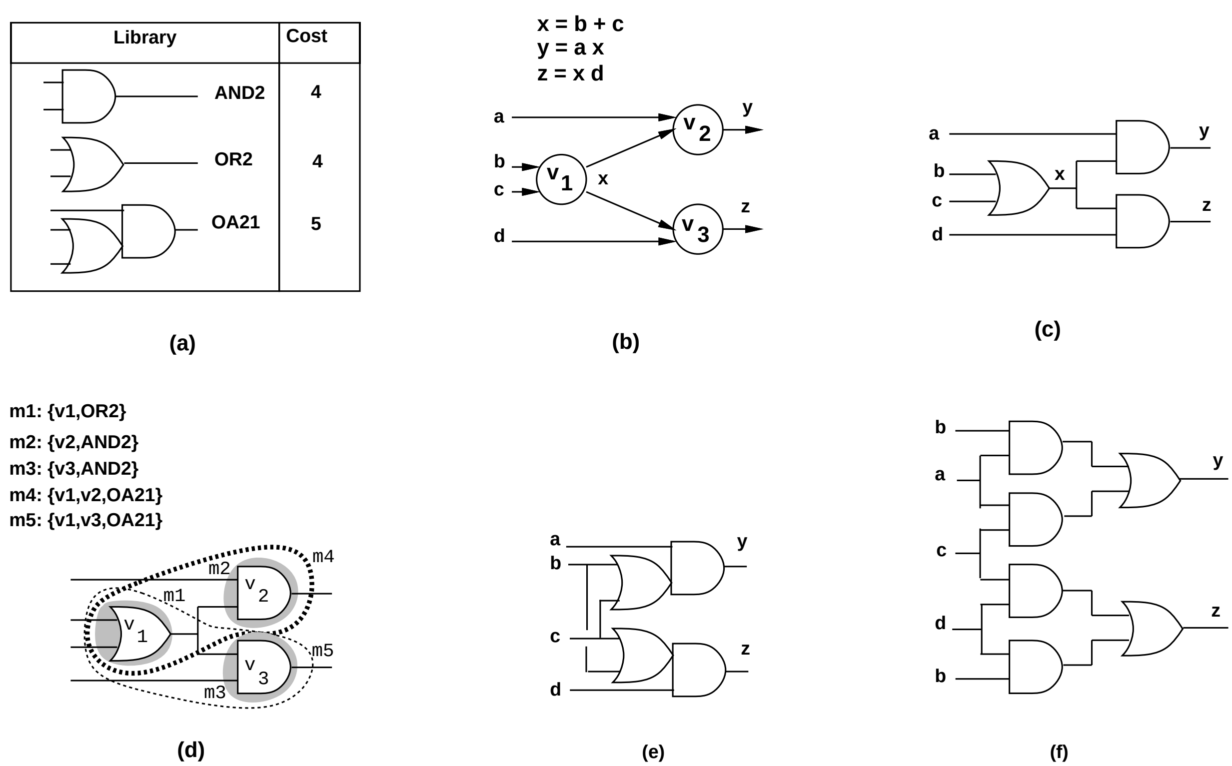
- Network granularity is fine.
  - Decomposition into *base* functions.
    - 2-input $AND, OR, NAND, NOR$.
- Trivial binding:
  - Replacement of each vertex by base cell.

Example



(a) (b) (c) (d)

Example


Library
Cost
AND2
4
OR2
4
OA21
5
(a)
x = b + c
y = a x
z = x d
a
b
c
d
v1
x
v2
v3
y
z
(b)
a
b
c
d
x
y
z
(c)
m1: {v1,OR2}
m2: {v2,AND2}
m3: {v3,AND2}
m4: {v1,v2,OA21}
m5: {v1,v3,OA21}
m1
m2
m3
m4
m5
v1
v2
v3
(d)
a
b
c
d
y
z
(e)
b
a
c
d
b
y
z
(f)

Example

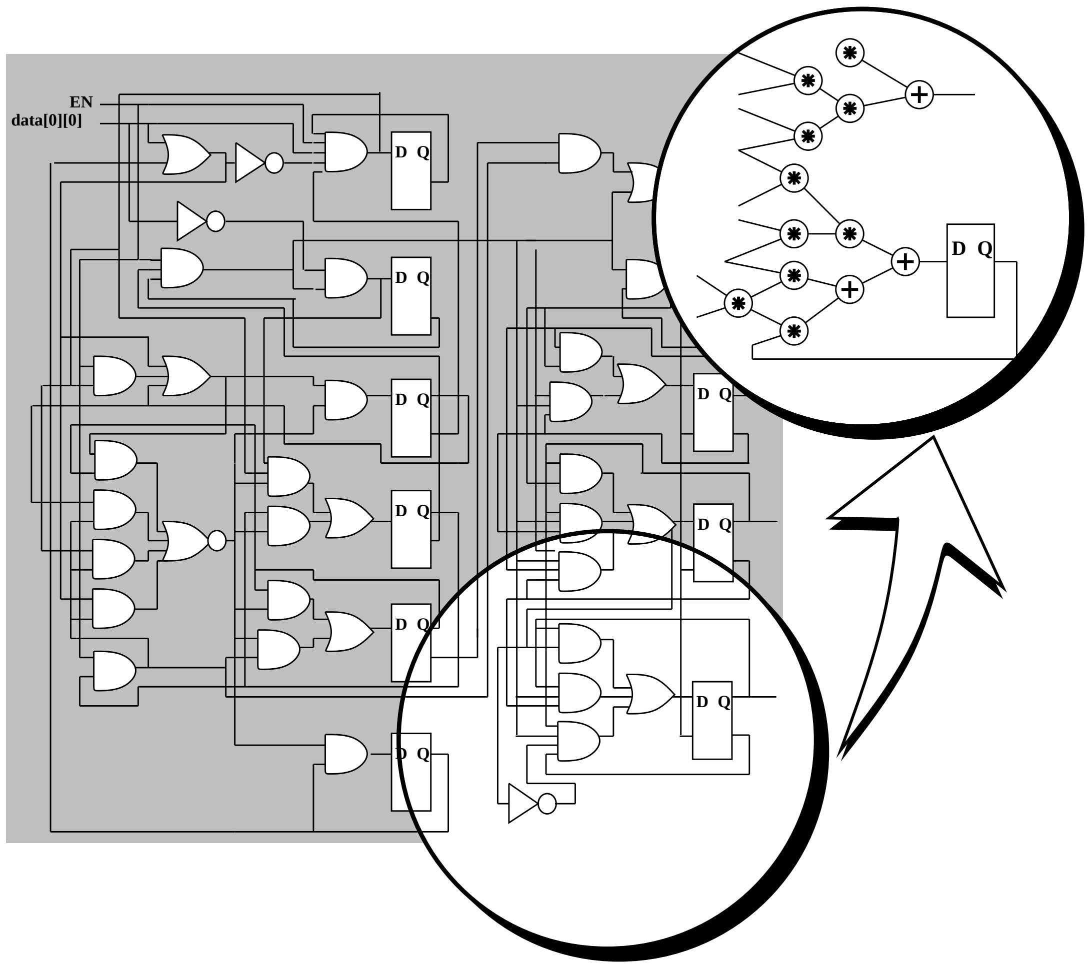
- Vertex covering:
  - Covering v_1: $(m_1 + m_4 + m_5)$.
  - Covering v_2: $(m_2 + m_4)$.
  - Covering v_3: $(m_3 + m_5)$.
- Input compatibility:
  - Match m_2 requires m_1:
    * $(m'_2 + m_1)$.
  - Match m_3 requires m_1:
    * $(m'_3 + m_1)$.
- Overall *binate* clause:
  - $(m_1 + m_4 + m_5)(m_2 + m_4)(m_3 + m_5)(m'_2 + m_1)(m'_3 + m_1) = 1$

Heuristic algorithms

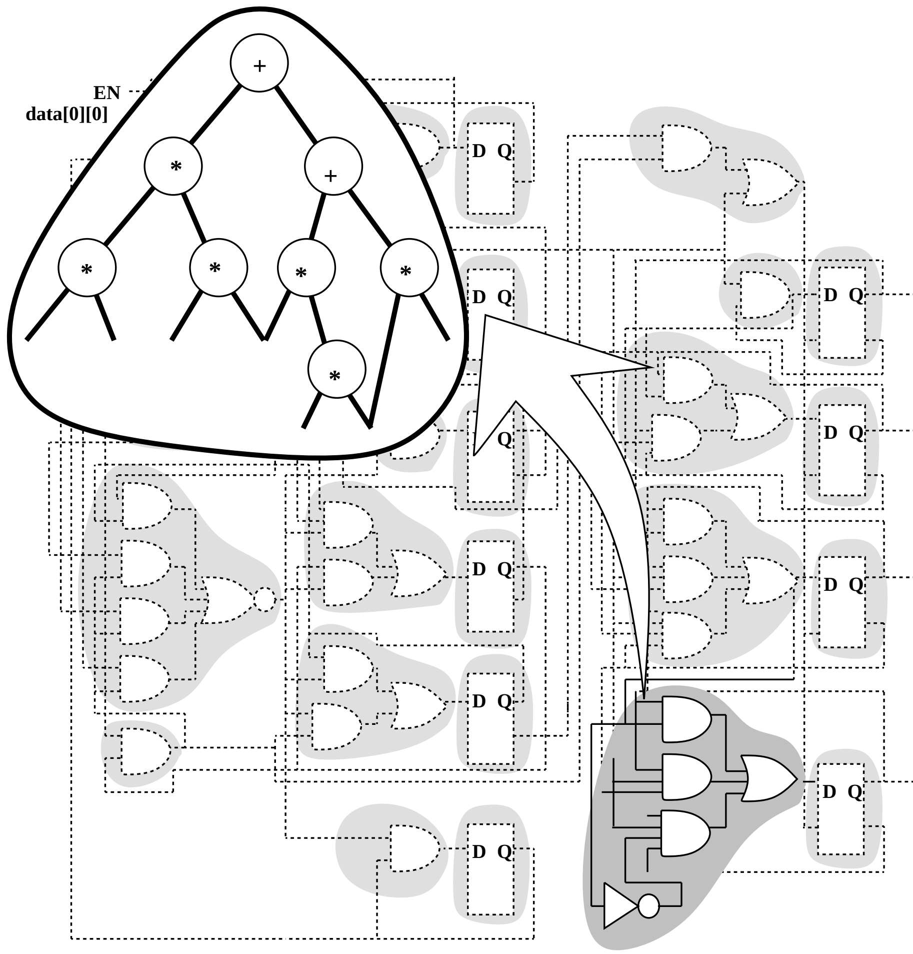
- Decomposition:
  - Cast network and library in standard form.
  - Decompose into *base functions*.
  - Example: NAND2 and INV.
- Partitioning:
  - Break network into *cones*.
  - Reduce to many multi-input single-output subnetworks.
- Covering:
  - Cover each subnetwork by library cells.

Decomposition

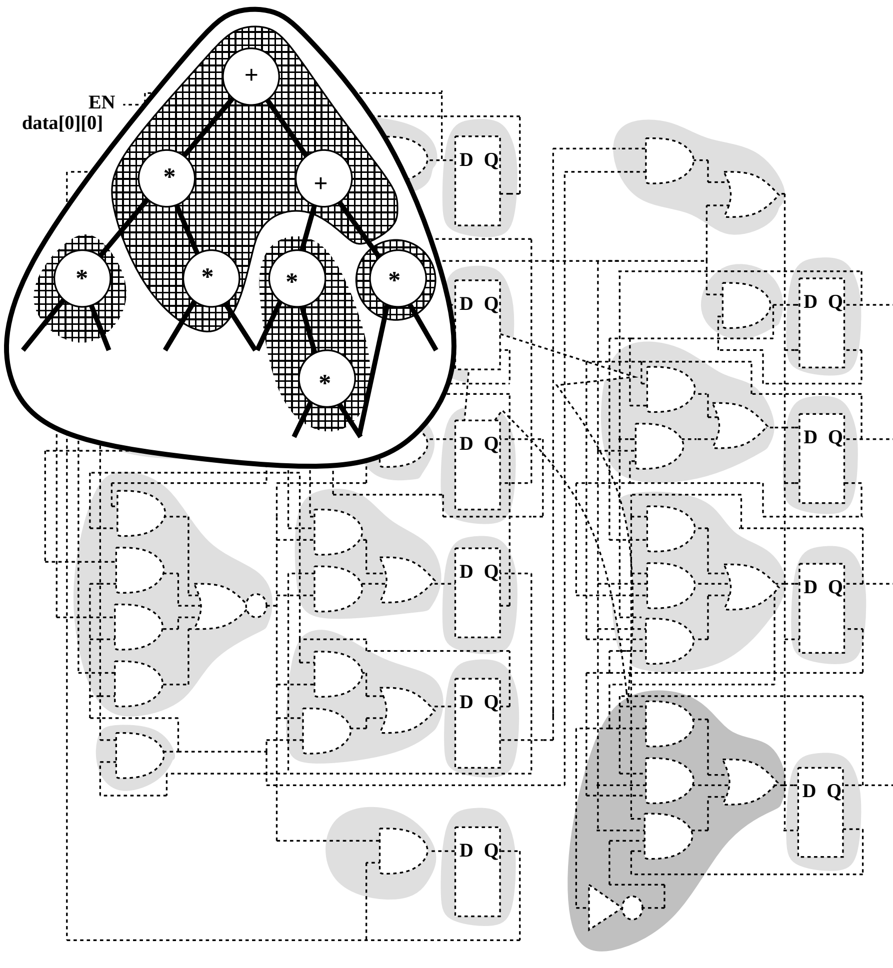
© GDM



Partitioning



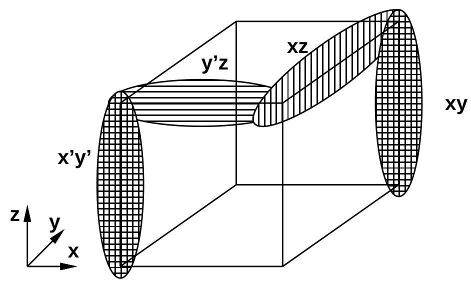
Covering



Heuristic algorithms

- Structural approach:
  - Model functions by *patterns*.
    - Example: trees, dags.
  - Rely on *pattern matching* techniques.
- Boolean approach:
  - Use Boolean models.
  - Solve *tautology* problem.
  - More powerful.

Example
Boolean versus structural matching



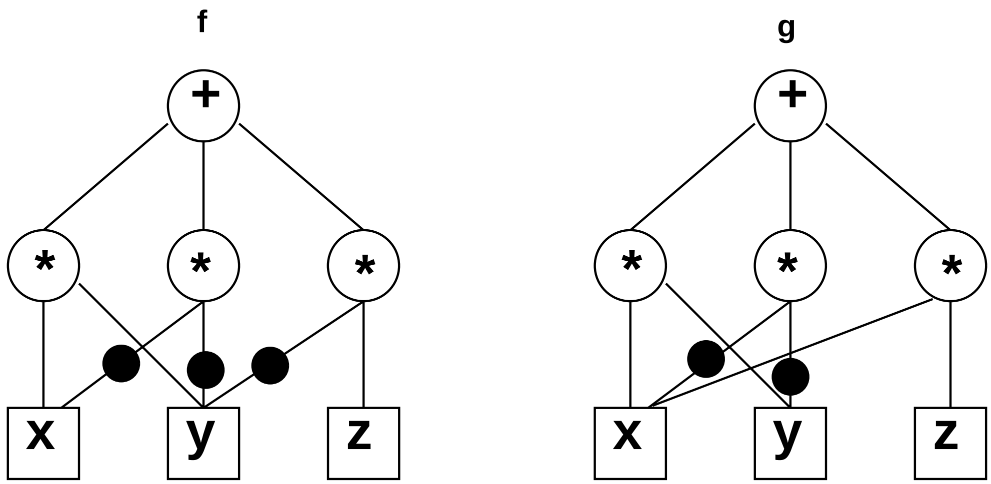
- $f = xy + x'y' + y'z$

- $g = xy + x'y' + xz$

- Function equality is a tautology:
  - Boolean match.

- Patterns may be different:
  - Structural match may not be found.

Example
Boolean versus structural matching



- $f = xy + x'y' + y'z$

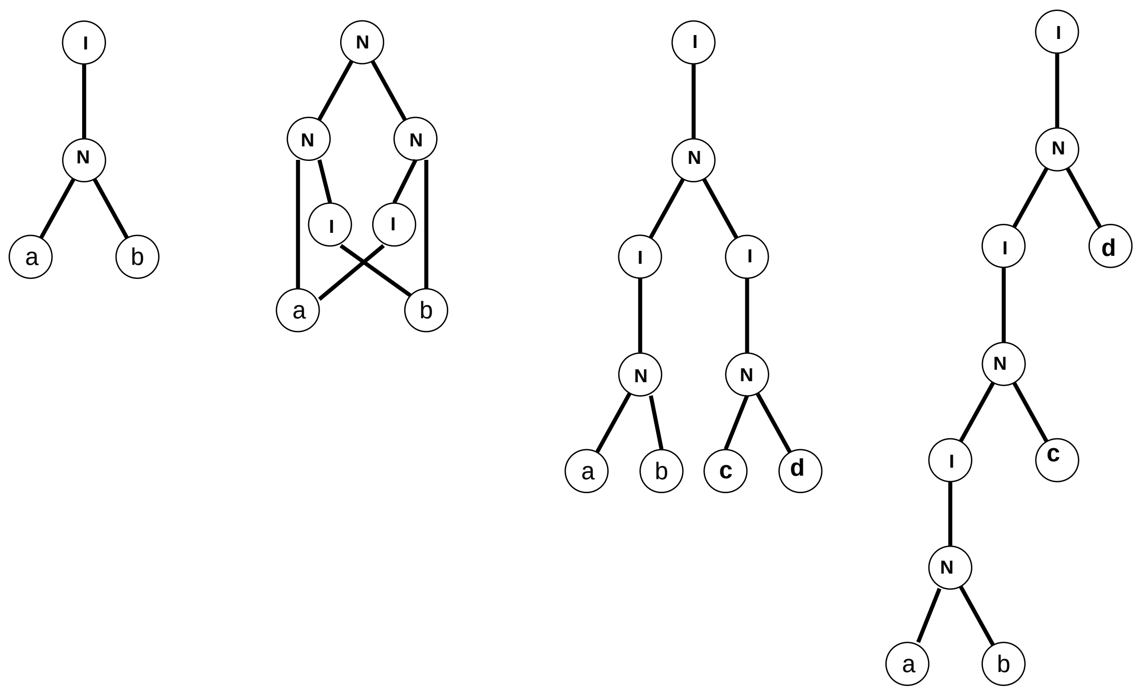
- $g = xy + x'y' + xz$

- Patterns do not match.

Structural matching and covering

- Expression patterns:
  - Represented by dags.
- Identify pattern dags in network:
  - Sub-graph isomorphism.
- Simplification:
  - Use tree patterns.

Example

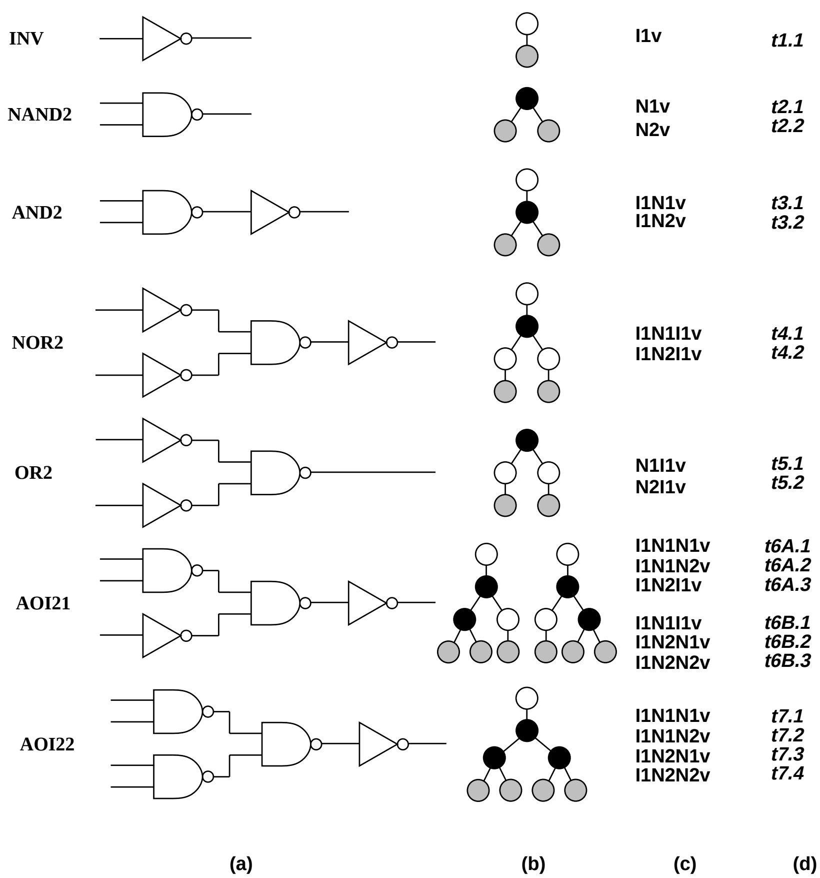


(a) (b) (c) (d)

Tree-based matching

- Network:
  - Partitioned and decomposed:
    * NOR2 (or NAND2) + INV.
    * Generic base functions.
  - *Subject tree.*
- Library:
  - Represented by trees.
  - Possibly more than one tree per cell.
- Pattern recognition:
  - Simple binary tree match.
  - Aho-Corasick automaton.

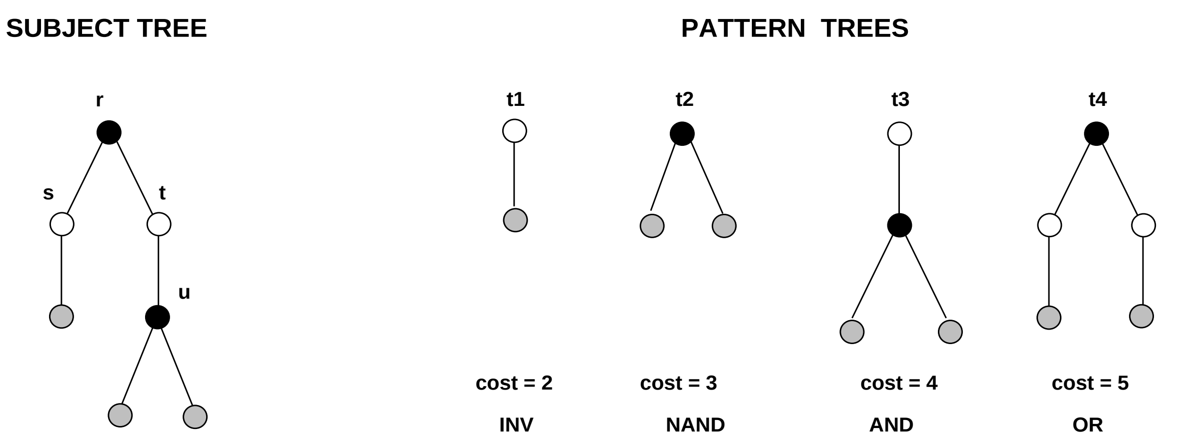
Simple library



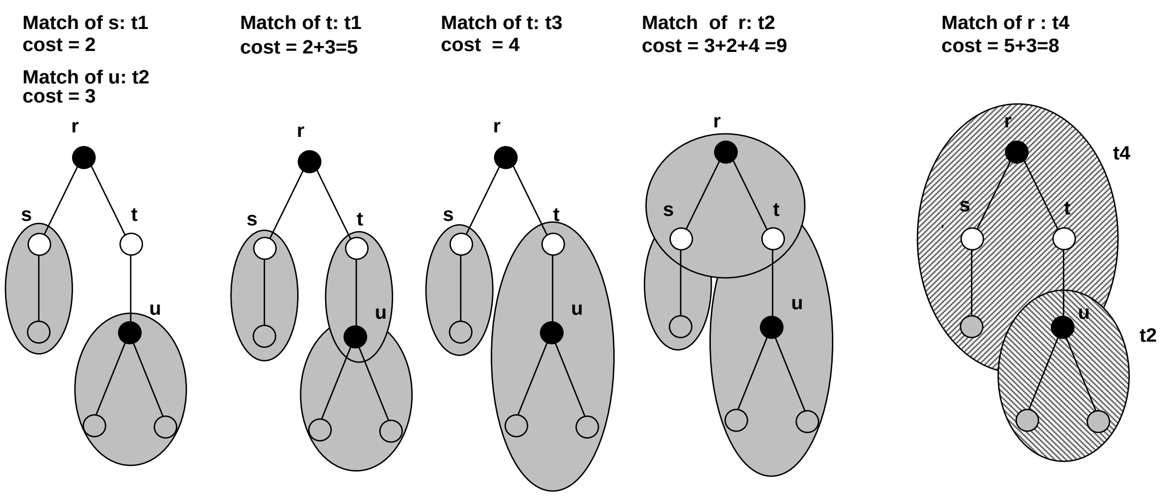
Tree covering

- Dynamic programming:
  - Visit subject tree bottom-up.
- At each vertex:
  - Attempt to match:
    - Locally rooted subtree.
    - All library cells.
- *Optimum* solution, for the subtree.

Example


SUBJECT TREE
r
s
t
u
PATTERN TREES
t1
t2
t3
t4
cost = 2
cost = 3
cost = 4
cost = 5
INV
NAND
AND
OR

Example


Match of s: t1
cost = 2
Match of u: t2
cost = 3
r
s
t
u
Match of t: t1
cost = 2+3=5
r
s
t
u
Match of t: t3
cost = 4
r
s
t
u
Match of r: t2
cost = 3+2+4 =9
r
s
t
u
Match of r : t4
cost = 5+3=8
r
s
t
u
t4
t2

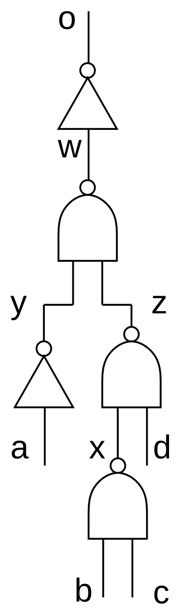
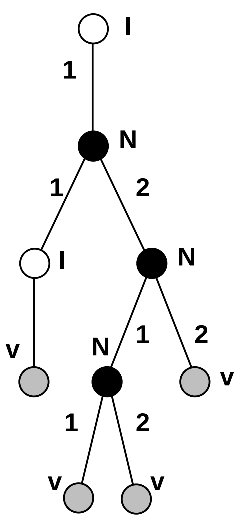
Example

- Minimum-area cover.

- Area costs:
  - INV:2; NAND2:3; AND2:4; AOI21:6.

- Best choice:
  - AOI21 fed by a NAND2 gate.

Example

Network	Subject graph	Vertex	Match	Gate	Cost
		x	*t2*	**NAND2(b,c)**	**3**
		y	*t1*	**INV(a)**	**2**
		z	*t2*	**NAND2(x,d)**	**2* 3 = 6**
		w	*t2*	**NAND2(y,z)**	**3 * 3 + 2 = 11**
		o	*t1*	**INV(w)**	**3 * 3 + 2 * 2 = 13**
			t3	**AND2(y,z)**	**2 * 3 + 4 + 2 = 12**
			t6B	**AOI21(x,d,a)**	**3 + 6 = 9**

Minimum delay cover

- Dynamic programming approach.

- Cost related to gate delay.

- Delay modeling:
  - Constant gate delay.
    - Straightforward.
  - Load-dependent delay:
    - Load fanout unknown.
    - Binning techniques.

Minimum delay cover
constant delays

- The cell pattern tree and the rooted subtree are isomorphic.
  - The vertex is labeled with the cell delay.

- The cell tree is isomorphic to a subtree with leaves L.
  - The vertex is labeled with the cell cost plus the *maximum* of the labels of L.

Example

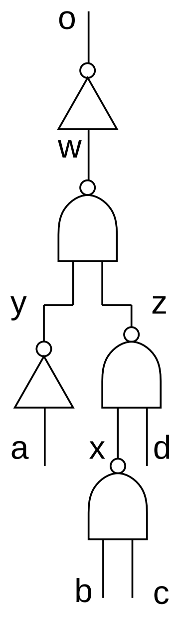
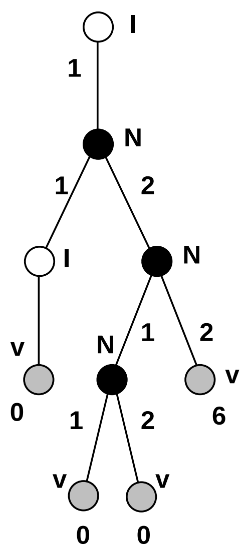
- Inputs data-ready times are 0 except for $t_d = 6$.

- Constant delays:
  - INV:2; NAND2:4; AND2:5; AOI21:10.

- Compute *data-ready* times bottom-up:
  - $t_x = 4, t_y = 2; t_z = 10 t_w = 14$.

- Best choice:
  - AND2, two NAND2 and an INV gate.

Example

<table>
<tr><th>Network</th><th>Subject graph</th><th>Vertex</th><th>Match</th><th>Gate</th><th>Cost</th></tr>
<tr><td rowspan="7"></td><td rowspan="7"></td><td>x</td><td>*t2*</td><td>**NAND2(b,c)**</td><td>**4**</td></tr>
<tr><td>y</td><td>*t1*</td><td>**INV(a)**</td><td>**2**</td></tr>
<tr><td>z</td><td>*t2*</td><td>**NAND2(x,d)**</td><td>**6 + 4 = 10**</td></tr>
<tr><td>w</td><td>*t2*</td><td>**NAND2(y,z)**</td><td>**10 + 4 = 14**</td></tr>
<tr><td rowspan="3">o</td><td>*t1*</td><td>**INV(w)**</td><td>**14 + 2 = 16**</td></tr>
<tr><td>*t3*</td><td>**AND2(y,z)**</td><td>**10 + 5 = 15**</td></tr>
<tr><td>*t6B*</td><td>**AOI21(x,d,a)**</td><td>**10 + 6 = 16**</td></tr>
</table>

Minimum delay cover
load-dependent delays

- Model:
  - Assume a finite set of load values.

- Dynamic programming approach:
  - Compute an array of solutions
    for each possible load.
  - For each input to a matching cell
    the best match for any load is selected.

- *Optimum* solution,
  when all possible loads are considered.

Example

- Inputs data-ready times are 0 except for $t_d = 6$.

- Load-dependent delays:

  - INV:1+l; NAND2:3+l; AND2:4+l; AOI21:9+l.

- Loads:

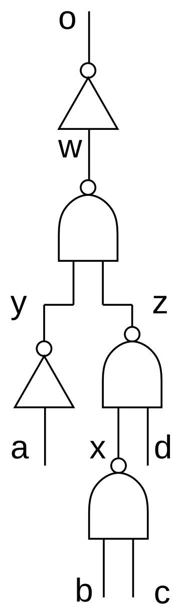
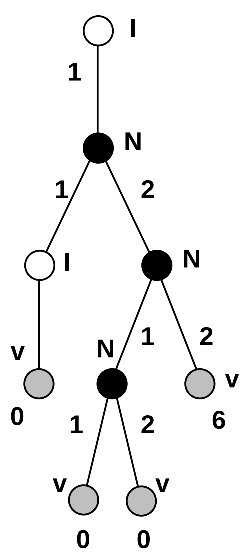
  - INV:1; NAND2:1; AND2:1; AOI21:1.

- Same solution as before.

Example

- Inputs data-ready times are 0 except for $t_d = 6$.
- Load-dependent delays:
  - INV:1+l; NAND2:3+l; AND2:4+l; AOI21:9+l; SINV:1+0.5l;.
- Loads:
  - INV:1; NAND2:1; AND2:1; AOI21:1; SINV:2.
- Assume output load is 1:
  - Same solution as before.
- Assume output load is 5:
  - Solution uses SINV cell.

Example

Network	Subject graph	Vertex	Match	Gate	Cost		
					Load=1	Load=2	Load=5
		x	*t2*	**NAND2(b,c)**	**4**	**5**	**8**
		y	*t1*	**INV(a)**	**2**	**3**	**6**
		z	*t2*	**NAND2(x,d)**	**10**	**11**	**14**
		w	*t2*	**NAND2(y,z)**	**14**	**15**	**18**
		o	*t1*	**INV(w)**			**20**
			t3	**AND2(y,z)**			**19**
			t6B	**AOI21(x,d,a)**			**20**
				SINV(w)			**18.5**

Library binding and polarity assignment

- Search for lower cost solution by not constraining the signal polarities.

- Most circuit allow us to choose the input/output signal polarities.

- Approaches:

  - Structural covering.

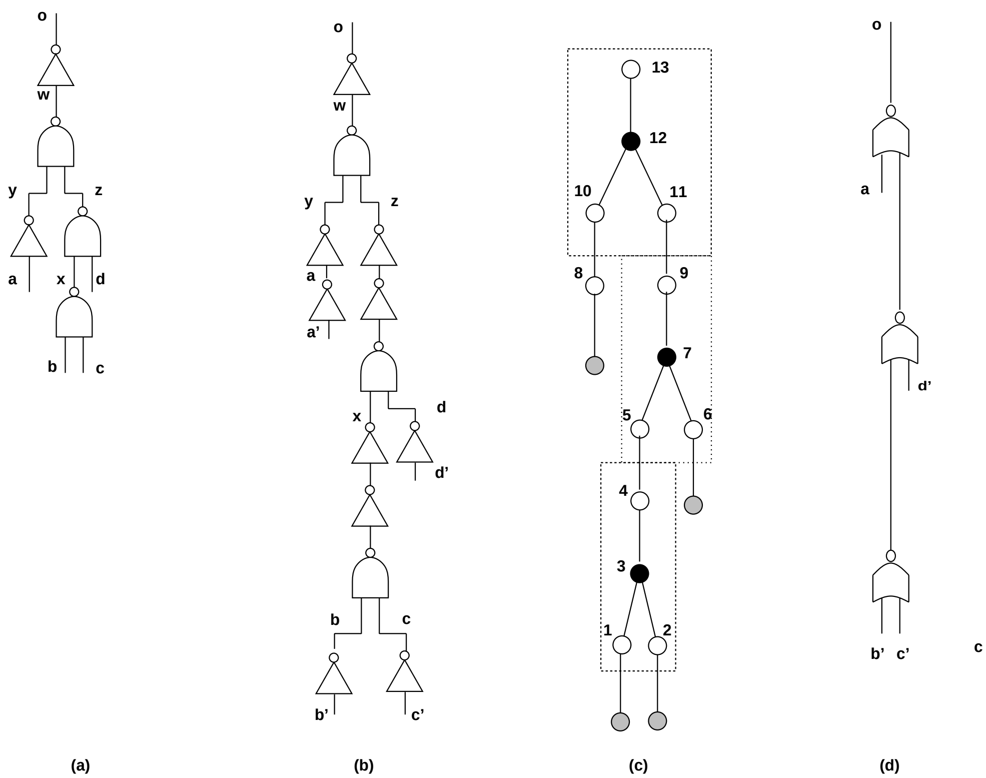
  - Boolean covering.

Structural covering and polarity assignment

- Pre-process subject network:
  - Add inverter pairs between NANDs.
  - Provide signals with both polarity.
- Add inverter-pair cell to the library:
  - To eliminate unneeded pairs.
  - Cell corresponds to a connection with zero cost.

Example

© GDM



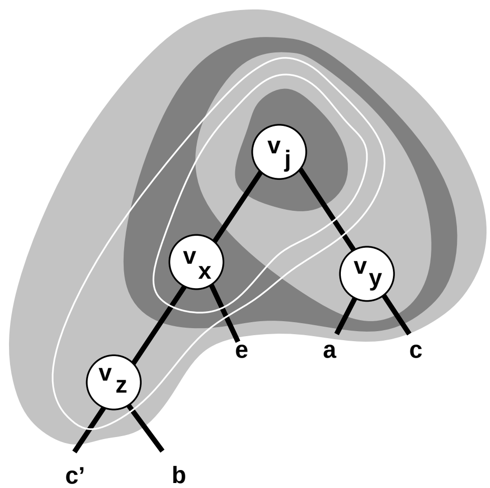
Boolean covering

- Decompose network into base functions.

- When considering vertex v_i:
  - Construct *clusters* by local elimination.
  - Several functions associated with v_i.

- Limit size and depth of clusters.

Example



$$
\begin{aligned}
f_{j,1} &= xy; \\
f_{j,2} &= x(a+c); \\
f_{j,3} &= (e+z)y; \\
f_{j,4} &= (e+z)(a+c); \\
f_{j,5} &= (e+c'+d)y; \\
f_{j,6} &= (e+c'+d)(a+c);
\end{aligned}
$$

Boolean matching
$\mathcal{P}$-equivalence

- *Cluster function* $f(\mathbf{x})$: sub-network behavior.

- *Pattern function* $g(\mathbf{y})$: cell behavior.

- $\mathcal{P}$-equivalence:

  – Exists a permutation operator $\mathcal{P}$, such that $f(\mathbf{x}) = g(\mathcal{P}\ \mathbf{x})$ is a tautology?

- Approaches:

  – Tautology check over all input permutations.

  – Multi-rooted pattern ROBDD capturing all permutations.

Input/output polarity assignment

- Allow for reassignment of input/output polarity.

- $\mathcal{NPN}$ classification of Boolean functions.

- $\mathcal{NPN}$-equivalence:

  – Exists a permutation matrix $\mathcal{P}$,
  and complementation operators $\mathcal{N}_i, \mathcal{N}_o$
  such that $f(\mathbf{x}) = \mathcal{N}_o \; g(\mathcal{P} \; \mathcal{N}_i \; \mathbf{x})$
  is a tautology?

- Variations:

  – $\mathcal{N}$-equivalence, $\mathcal{PN}$-equivalence

Boolean matching

- *Pin assignment* problem.
  - Map cluster variables $\mathbf{x}$ to pattern vars $\mathbf{y}$.
  - Characteristic equation: $\mathcal{A}(\mathbf{x}, \mathbf{y}) = 1$.

- Pattern function under variable assignment:
  - $g_{\mathcal{A}}(\mathbf{x}) = \mathcal{S}_{\mathbf{y}} \mathcal{A}(\mathbf{x}, \mathbf{y}) \; g(\mathbf{y})$

- *Tautology problem*.
  - $f(\mathbf{x}) \; \overline{\oplus} \; g_{\mathcal{A}}(\mathbf{x})$
  - $\forall_{\mathbf{x}}(f(\mathbf{x}) \; \overline{\oplus} \; \mathcal{S}_{\mathbf{y}} \; (\mathcal{A}(\mathbf{x}, \mathbf{y}) \; g(\mathbf{y})))$

Example

- Assign x_1 to y_2' and x_2 to y_1.

- Characteristic equation:
  - $A(x_1, x_2, y_1, y_2) = (x_1 \oplus y_2)(x_2 \overline{\oplus}\ y_1)$

- AND pattern function:
  - $g = y_1 y_2$

- Pattern function under assignment:
  - $\mathcal{S}_{y_1,y_2}\mathcal{A}g =$
    $= \mathcal{S}_{y_1,y_2}(x_1 \oplus y_2)(x_2 \overline{\oplus} y_1)y_1 y_2 = x_2 x_1'$

Signatures and filters

- Capture some properties of Boolean functions.

- If signatures do not match, there is no match.

- Used as filters to reduce computation.

- Signatures:

  - Unateness.

  - Symmetries.

  - Co-factor sizes.

  - Spectra.

Filters based on unateness and symmetries

- Any pin assignment must associate
  - unate (binate) variables in $f(\mathbf{x})$ with unate (binate) variables in $g(\mathbf{y})$.

- Variables or groups of variables
  - that are interchangeable in $f(\mathbf{x})$ must be interchangeable in $g(\mathbf{y})$.

Example

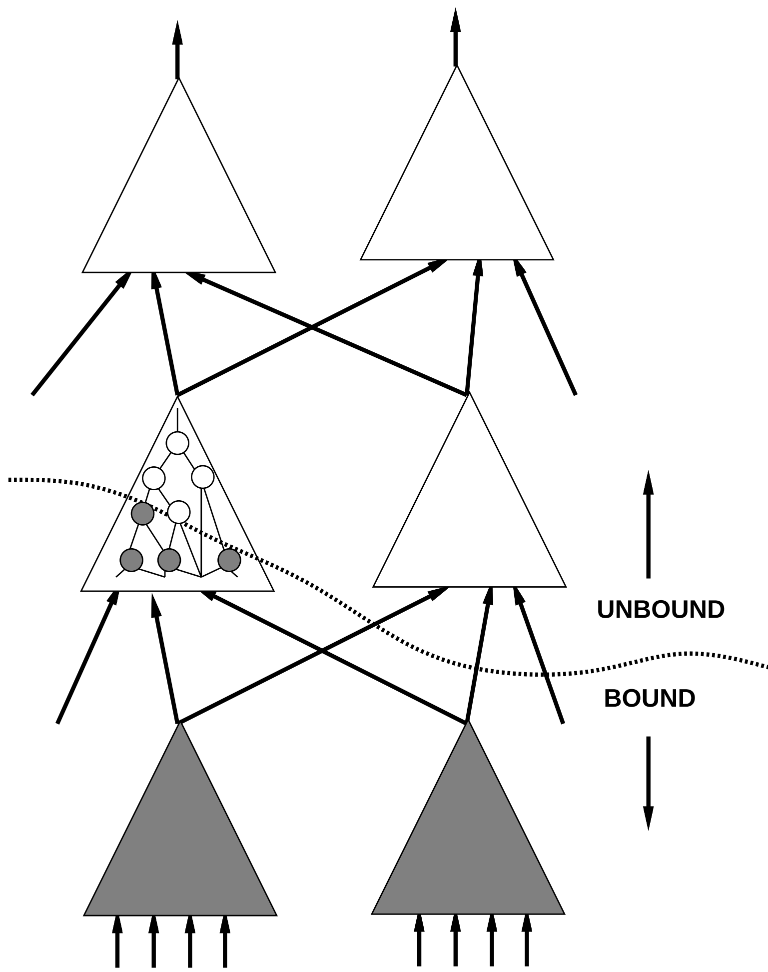
- Cluster function: $f = abc$.
  - Symmetries:$\{(a, b, c)\}$ – unate.

- Pattern functions:
  - $g_1 = a + b + c$
    - Symmetries:$\{(a, b, c)\}$ – unate.
  - $g_2 = ab + c$
    - Symmetries:$\{(a, b)(c)\}$ – unate.
  - $g_3 = abc' + a'b'c$
    - Symmetries:$\{(a, b, c)\}$ – binate.

Concurrent optimization and library binding

- Motivation:
  - Logic simplification is usually done prior to binding.
  - Logic simplification/substitution can be combined with binding.
- Mechanism:
  - Binding induces some *don't care* conditions.
  - Exploit *don't cares* as degrees of freedom in matching.

Example



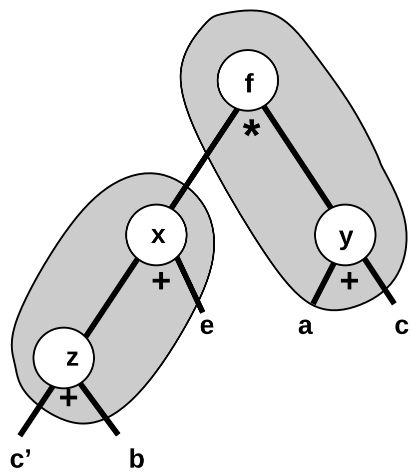
Boolean matching with *don't care* conditions

- Given $f(\mathbf{x}), f_{DC}(\mathbf{x})$ and $g(\mathbf{y})$:
  - g matches f if g is equivalent to $\tilde{f}$ where $f \cdot f'_{DC} \leq \tilde{f} \leq f + f_{DC}$

- Matching condition:
  - $\forall_{\mathbf{x}}(f_{DC}(\mathbf{x}) + f(\mathbf{x}) \,\overline{\oplus}\, \mathcal{S}_{\mathbf{y}}\, (\mathcal{A}(\mathbf{x}, \mathbf{y})\; g(\mathbf{y})))$

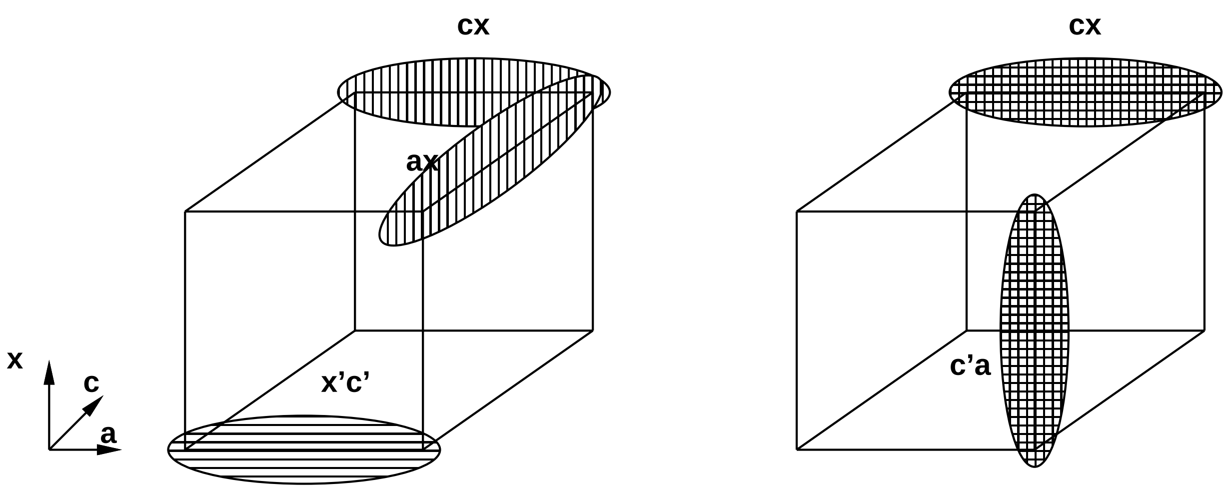
Example

© GDM

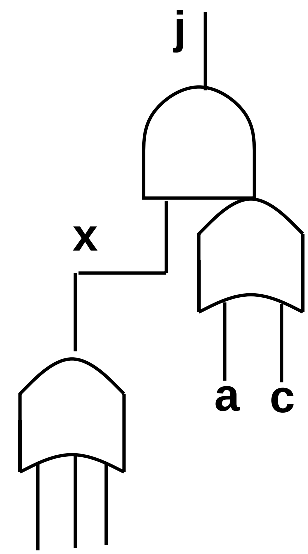


- Assume v_x is bound to $OR3(c', b, e)$.
- *Don't care* set includes $x \oplus (c' + b + e)$.
- Consider $f_j = x(a + c)$ with $CDC = x'c'$.
- No simplification. Mapping into AOI gate.
- Matching with DC. Mapping into MUX gate.

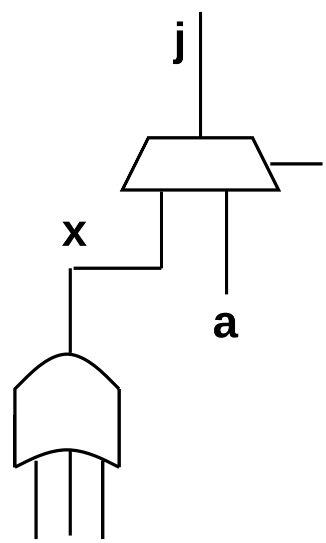
Example



Example



(a)



(b)

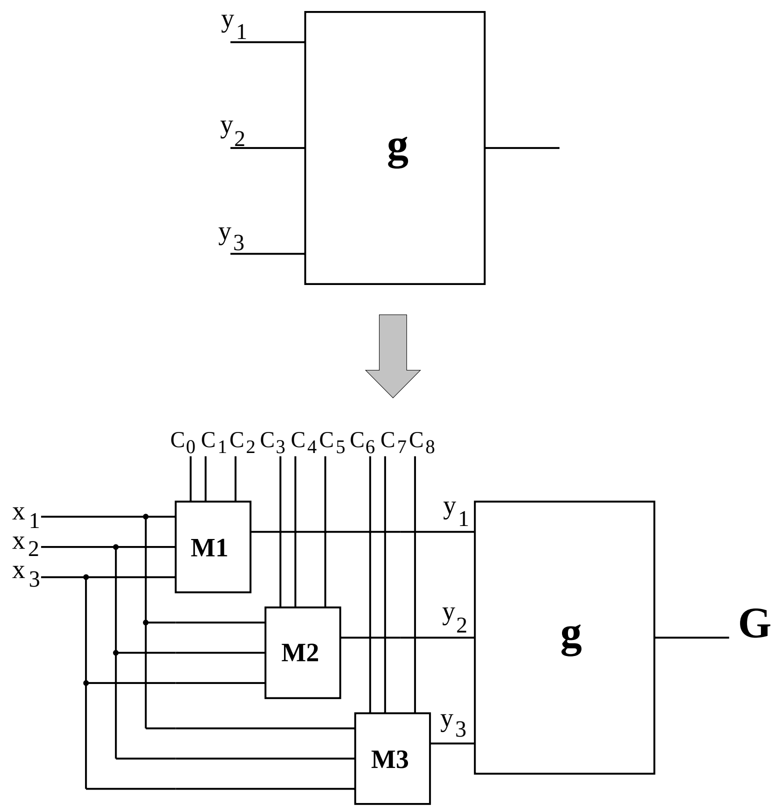
Extended matching

- Augment pattern function with mux function.
  - Each cell input can be routed to any cluster input (or voltage rail).
  - Input polarity can be changed.
  - Cell and cluster may differ input size.

- Define composite function $G(\mathbf{x}, \mathbf{c})$:
  - Pin assignment is determining $\mathbf{c}$.

- Matching formula: $M(\mathbf{c}) = \forall_{\mathbf{x}} [G(\mathbf{x}, \mathbf{c}) \overline{\oplus} f(\mathbf{x})]$

Example



- $g = y_1 + y_2 \ y_3'$

- $y_1(\mathbf{c}, \mathbf{x}) = (c_0 c_1 x_1 + c_0 c_1' x_2 + c_0' c_1 x_3) \oplus c_2$

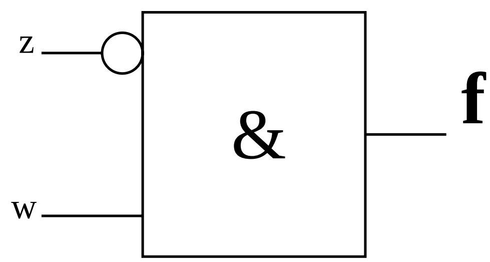
- $G = y_1(\mathbf{c}, \mathbf{x}) + y_2(\mathbf{c}, \mathbf{x}) \ y_3(\mathbf{c}, \mathbf{x})'$

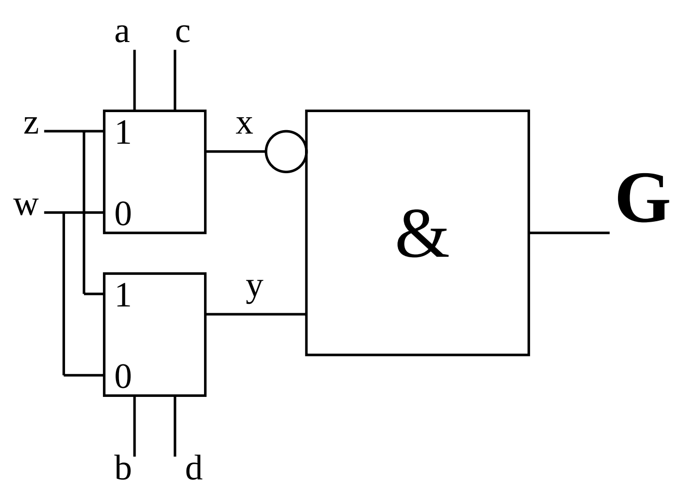
Extended matching modeling

- Model composite functions by ROBDDs.
  - Assume: n-input cluster and m-input cell.
  - For each cell input:
    - $\lceil log_2 n \rceil$ variables for pin permutation.
    - One variable for input polarity.
  - Total size of **c**: $m(\lceil log_2 n \rceil + 1)$.

- A match exists if there is at least one value of **c** satisfying $M(\mathbf{c}) = \forall_{\mathbf{x}} [G(\mathbf{x}, \mathbf{c}) \overline{\oplus} f(\mathbf{x})]$.

Example





- $g = x'y,\ f = wz'$
- $G(a,b,c,d,w,z) = (c \oplus (za + wa'))'(d \oplus (zb + wb'))$
- $f \overline{\oplus} G = (wz')\overline{\oplus}((c \oplus (za + wa'))'(d \oplus (zb + wb')))$
- $M(a,b,c,d) = ab'c'd' + a'bcd$

Extended matching

- Captures implicitly all possible matches.

- No extra burden when exploiting *don't care* sets.

  - $M(\mathbf{c}) = \forall_{\mathbf{x}} [G(\mathbf{x}, \mathbf{c}) \; \overline{\oplus} \; f(\mathbf{x}) + f_{DC}(\mathbf{x})]$

- Efficient BDD-based representation.

- Extensions to support multiple-output matching

Summary

- Library binding is very important.
- Rule-based approach:
  - General, sometimes inefficient.
- Algorithmic approach:
  - Pattern-based: fast, but limited.
  - Boolean: more general and efficient.