Libraries and Mapping

Virendra Singh
Indian Institute of Science
Bangalore

Courtesy: Giovanni De Micheli

Library Binding

- **Objective**
  - **Libraries. Problem formulation and analysis**
  - **Algorithms for library binding based on structural methods**

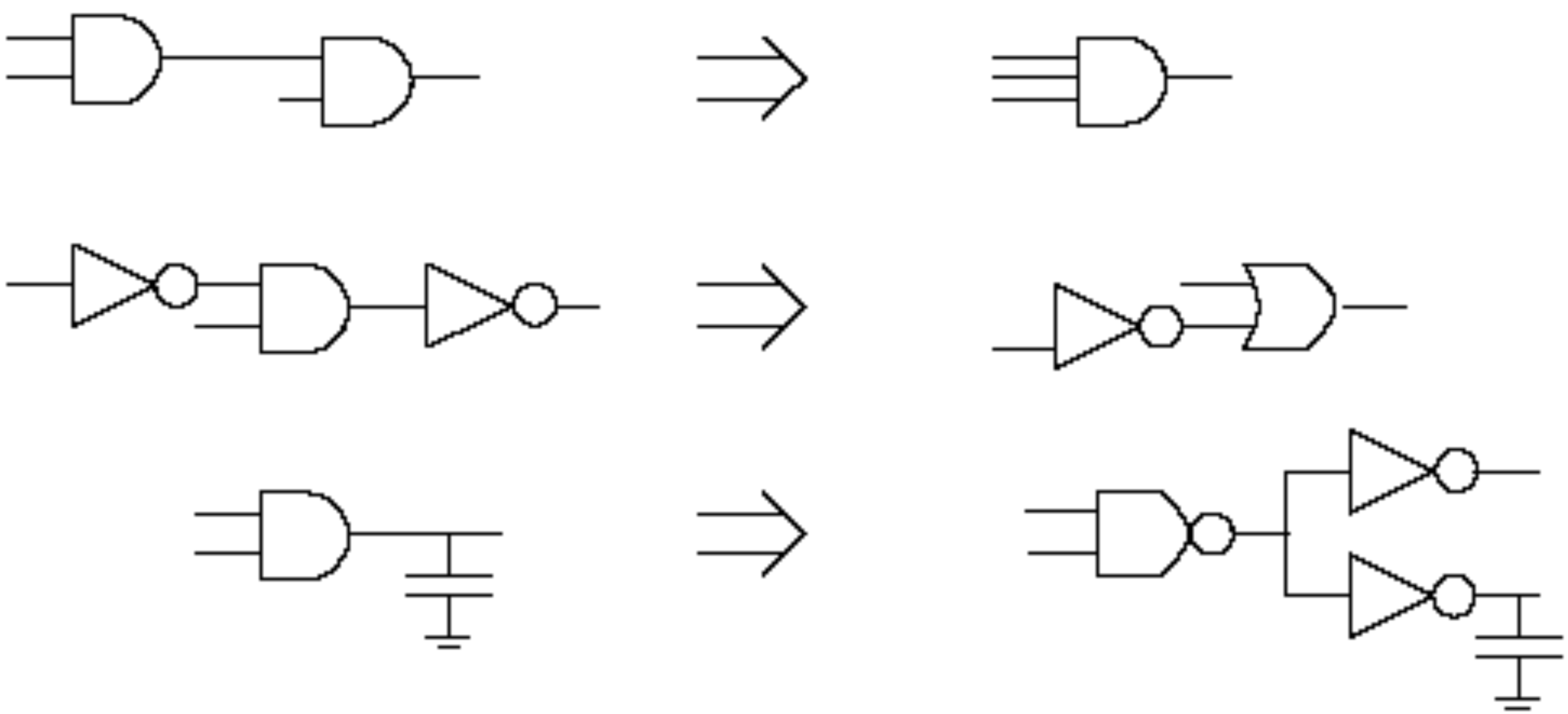
Library binding

- Given an unbound logic network and a set of library cells
  - Transform into an interconnection of instances of library cells
  - Optimize delay
    - (under area or power constraints)
  - Optimize area
    - Under delay and/or power constraints
  - Optimize power
    - Under delay and/or area constraints
- Library binding is called also technology mapping
  - Redesigning circuits in different technologies

Major approaches

- Rule-based systems
  - Generic, handle all types of cells and situations
  - Hard to obtain circuit with specific properties
  - Data base:
    - Set of pattern pairs
    - Local search: detect pattern, implement its best realization
- Heuristic algorithms
  - Typically restricted to single-output combinational cells
  - Library described by cell functionality and parameters
- Most systems use a combination of both approaches:
  - Rules are used for I/Os, high buffering requirements, …

Example



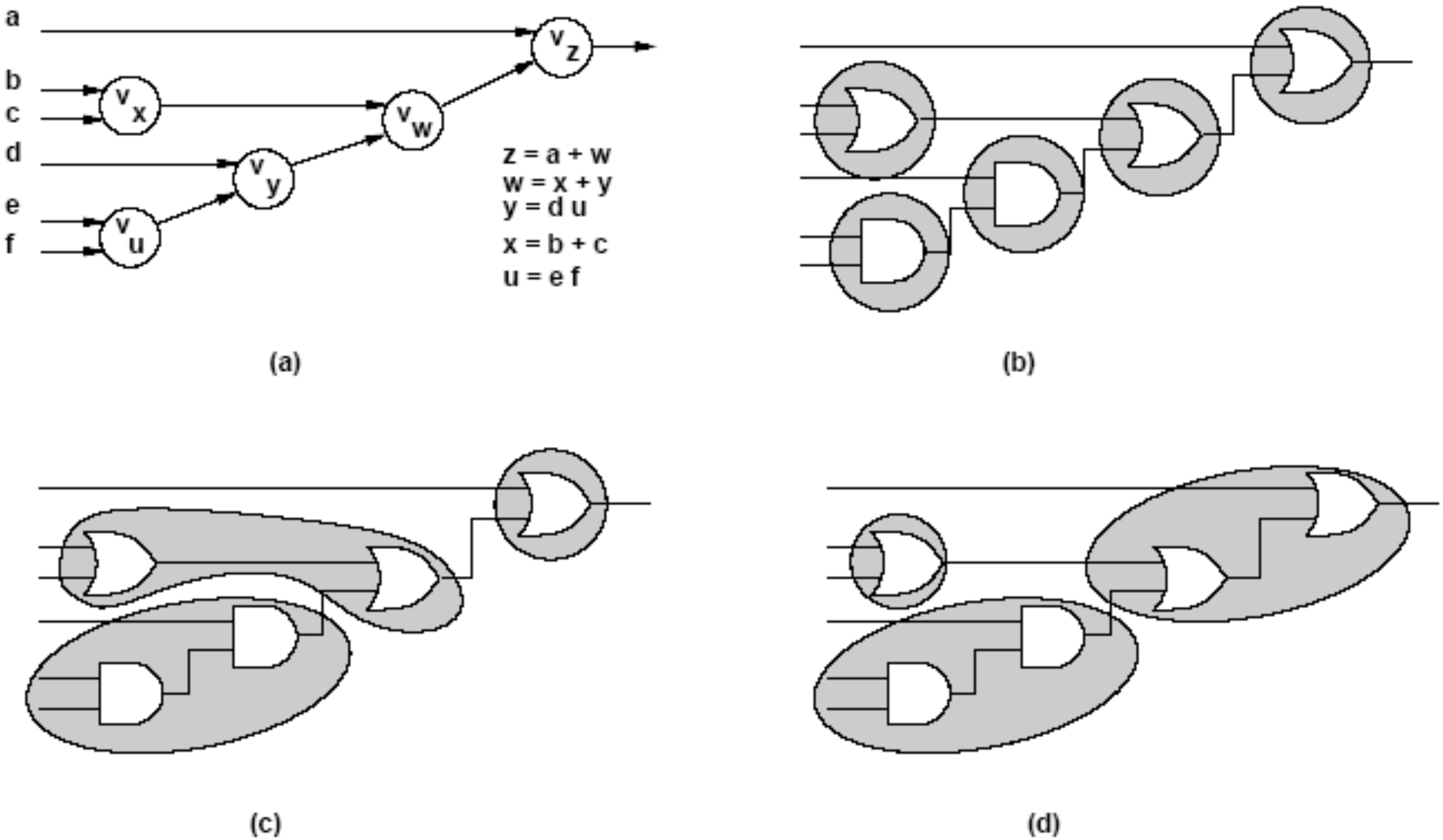
Library binding: issues

- **Matching**:
  - A cell matches a sub-network when their terminal behavior is the same
  - Tautology problem
  - *Input-variable* assignment problem
- **Covering**:
  - A cover of an unbound network is a partition into sub-networks which can be replaced by library cells.
  - Binate covering problem

Assumptions

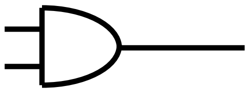
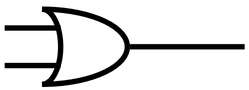
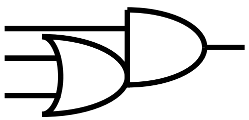
- Network granularity is fine
  - Decomposition into base functions:
  - 2-input AND, OR, NAND, NOR
- Trivial binding
  - Use base cells to realize decomposed network
  - There exists always a trivial binding:
    - Base-cost solution…

Example



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

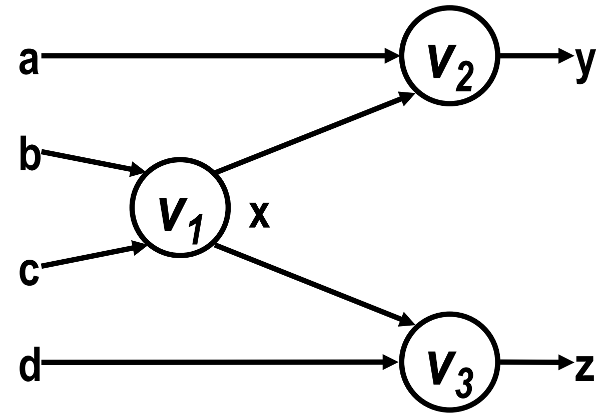
Example

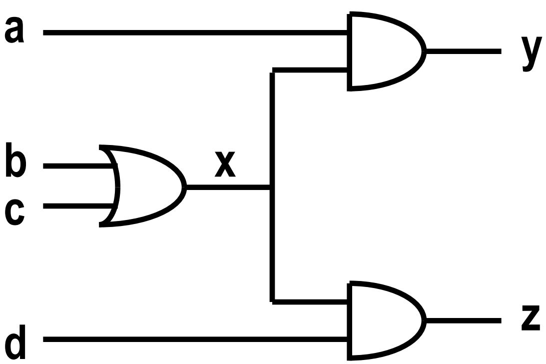
Library		Cost
	AND2	4
	OR2	4
	OA21	5

$x = b + c$

$y = ax$

$z = xd$





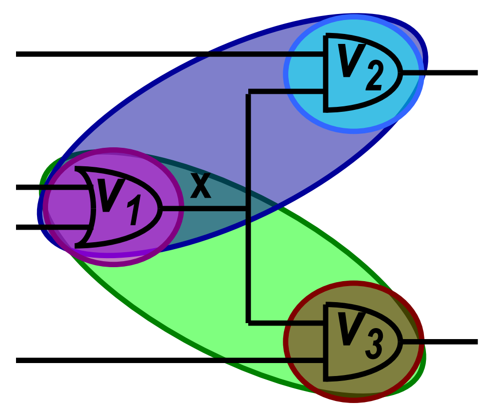
m_1: $\{v_1, \text{OR2}\}$

m_2: $\{v_2, \text{AND2}\}$

m_3: $\{v_3, \text{AND2}\}$

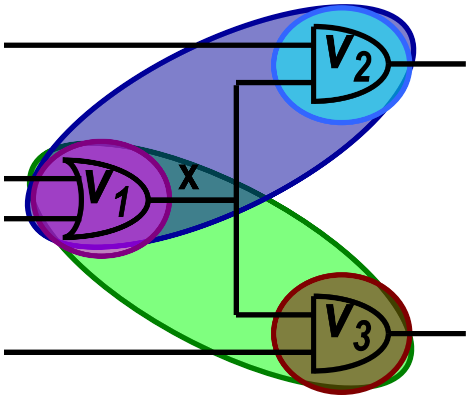
m_4: $\{v_1, v_2, \text{OA21}\}$

m_5: $\{v_1, v_3, \text{OA21}\}$



Example

- **Vertex covering:**
  - Covering v_1 : $(m_1 + m_4 + m_5)$
  - Covering v_2 : $(m_2 + m_4)$
  - Covering v_3 : $(m_3 + m_5)$

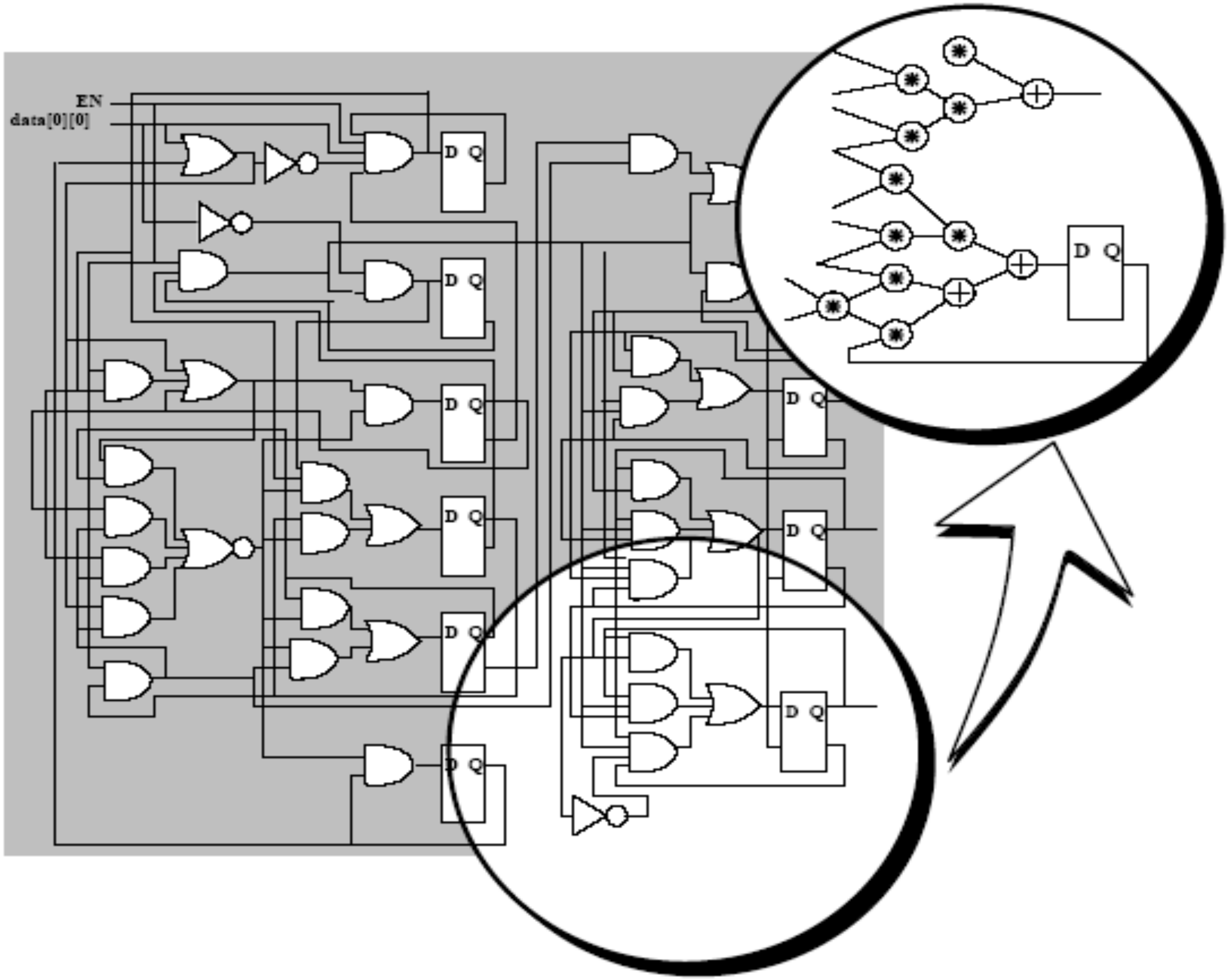


- **Input compatibility:**
  - Match m_2 requires m_1
    - $(m'_2 + m_1)$
  - Match m_3 requires m_1
    - $(m'_3 + m_1)$
- **Overall binate covering clause**
  - $(m_1+m_4+m_5) (m_2+m_4)(m_3+m_5)(m'_2+m_1)(m'_3+m_1) = 1$

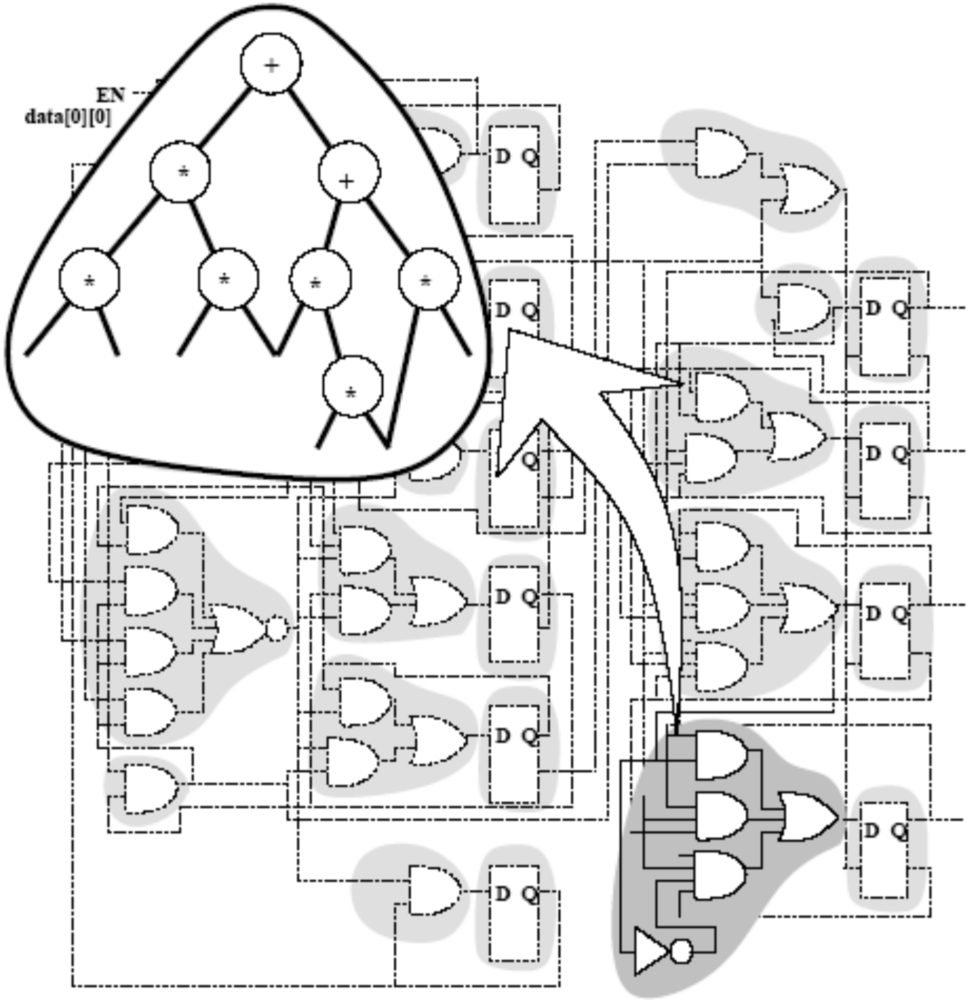
Heuristic approach to library binding

- **Split problem into various stages:**
  - 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 networks
  - Covering
    - Cover each sub-network by library cells
- **Most tools use this strategy**
  - Sometimes stages are merged

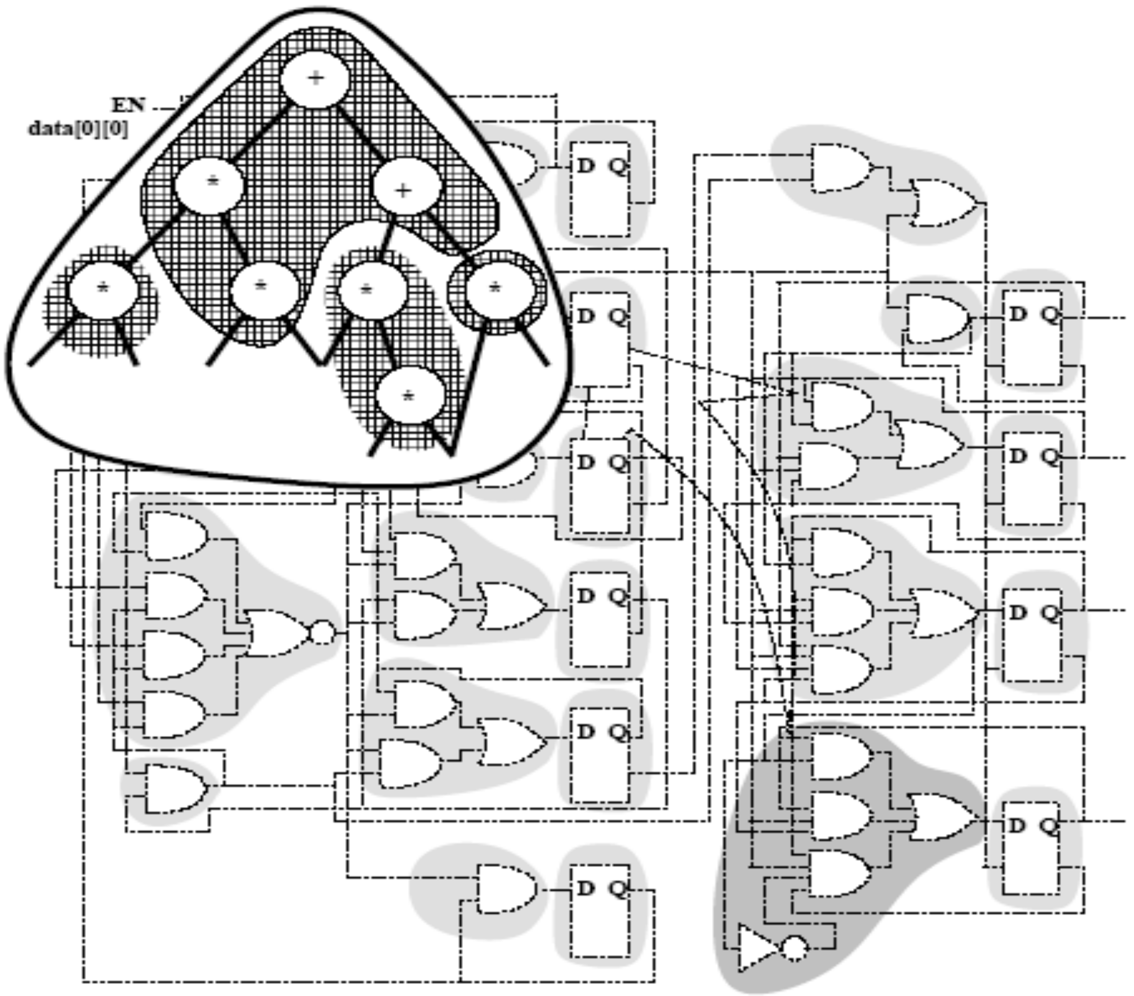
Decomposition



Partitioning



Covering

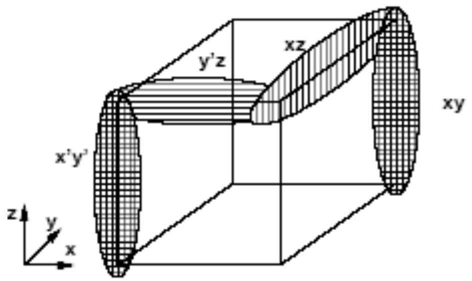


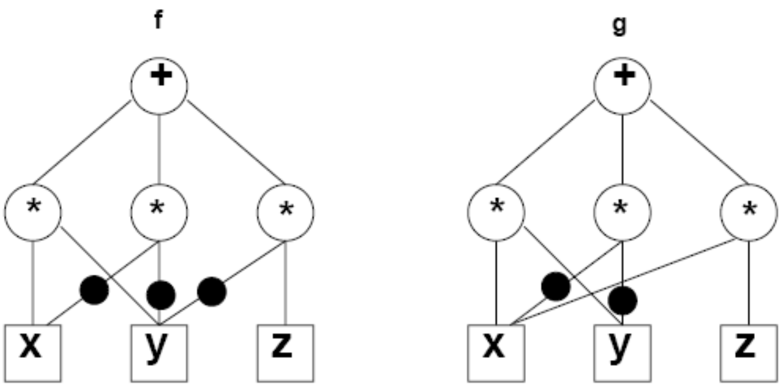
Heuristic algorithms

- Structural approach
  - Model functions by patterns
    - Example: tree, dags
  - Rely on pattern matching techniques
- Boolean approach
  - Use Boolean models
  - Solve the tautology problem
    - Use BDD technology
  - More powerful

Example

- **Boolean** vs. **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 exist

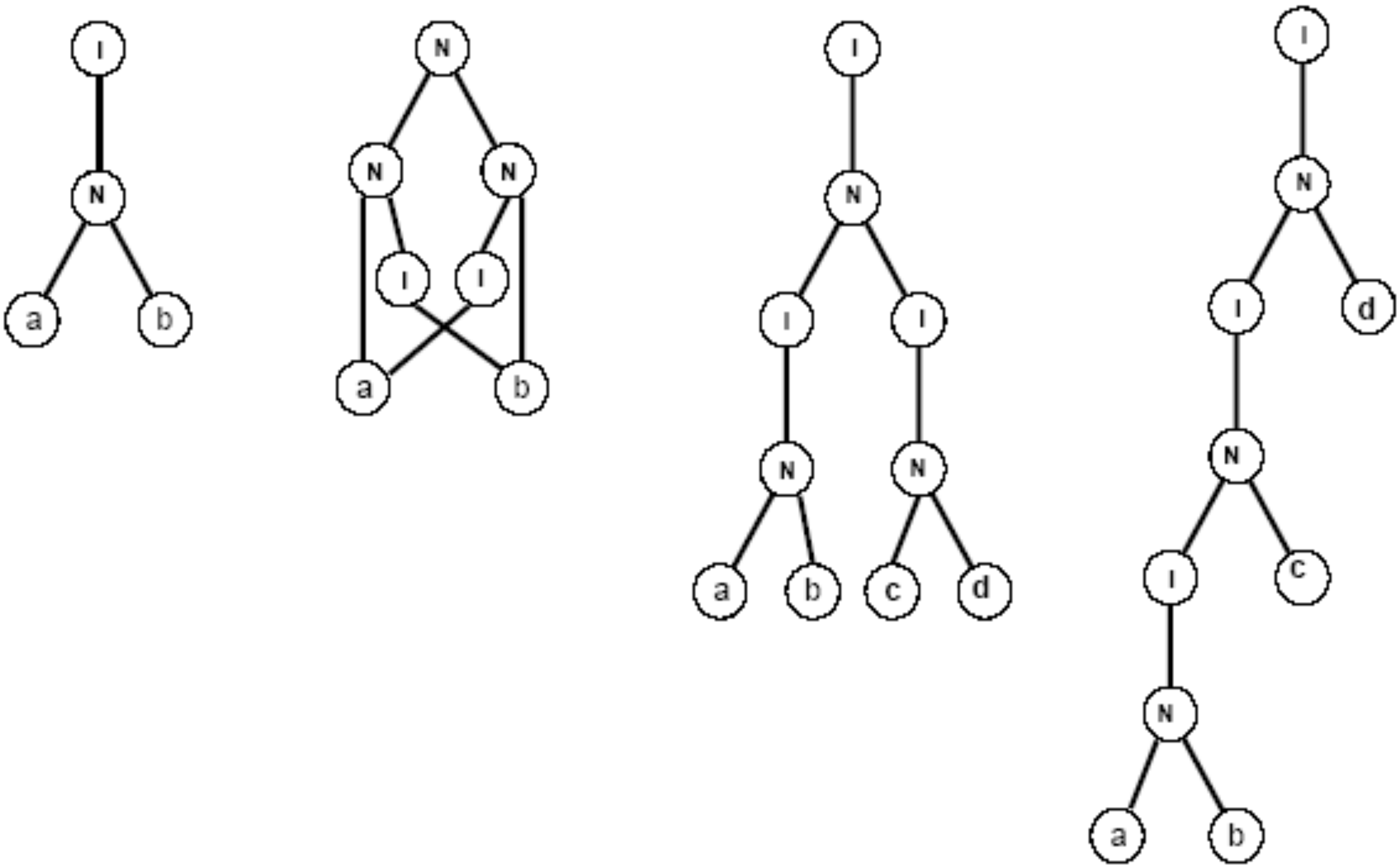




Structural matching and covering

- Expression patterns
  - Represented by dags
- Identify pattern dags in network
  - Sub-graph isomorphism
- Simplification:
  - Use tree patterns
- Typical problems with EXORs and MAJority functions

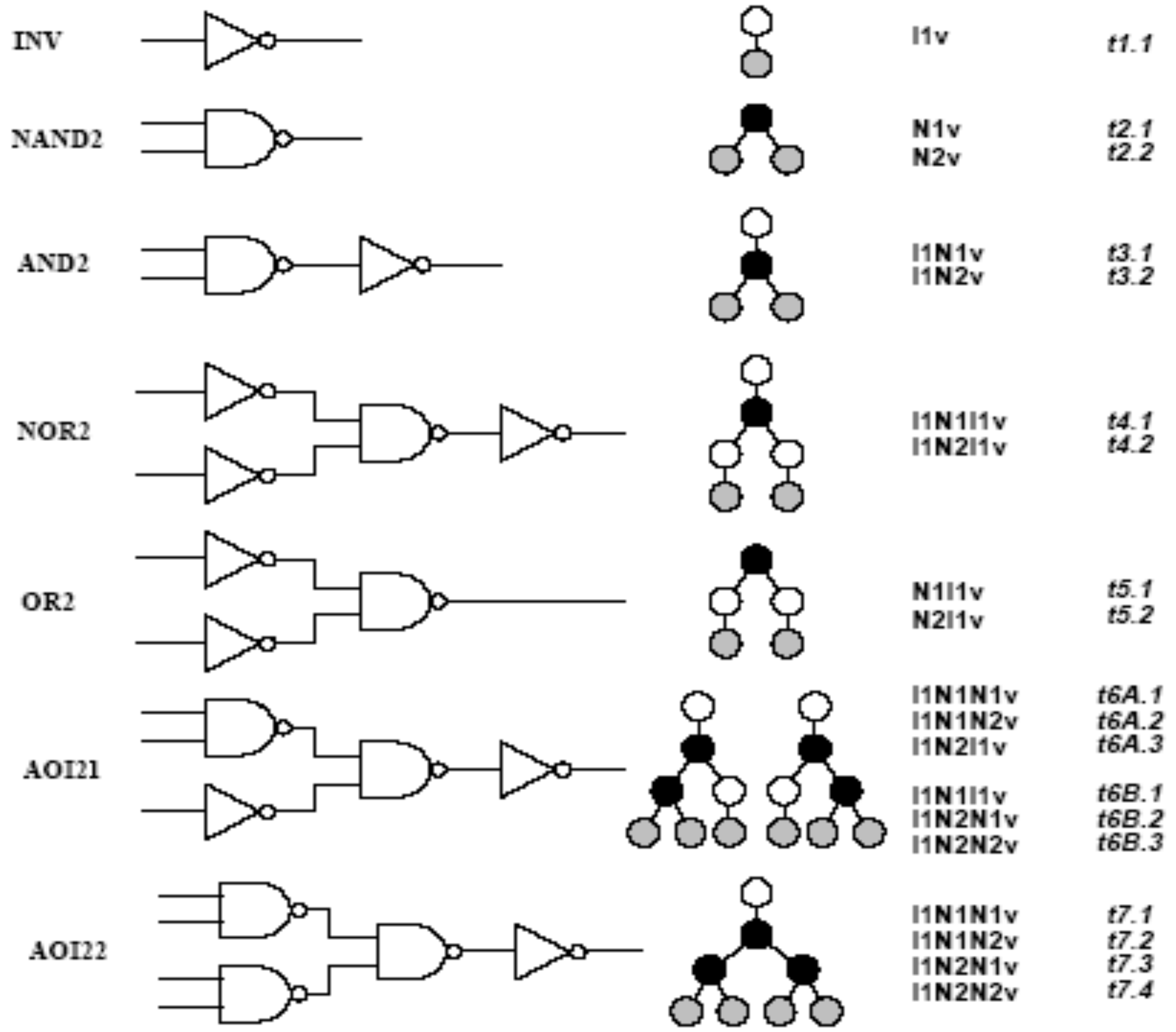
Example



Tree-based matching

- **Network:**
  - **Partitioned and decomposed**
    - **NOR2 (or NAND2) + INV**
    - **Generic base functions**
      - Not much used
    - **Subject tree**
- **Library**
  - **Represented by trees**
  - **Possibly more than one tree per cell**
- **Pattern recognition**
  - **Simple binary tree match**
  - **Aho-Corasik automaton**

Simple library

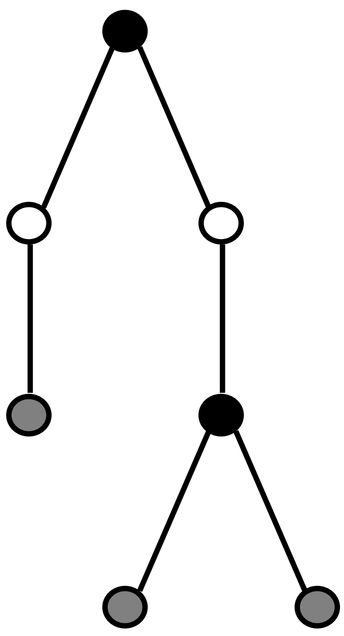


Tree covering

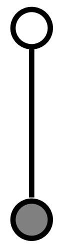
- Dynamic programming
  - Visit subject tree bottom up
- At each vertex
  - Attempt to match:
    - Locally rooted subtree to all library cell
    - Find best match and record
  - There is always a match when the base cells are in the library
- Bottom-up search yields and optimum cover
- Caveat:
  - Mapping into trees is a distortion for some cells
  - Overall optimality is weakened by the overall strategy of splitting into several stages

Example

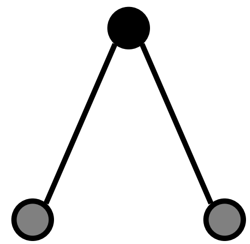
SUBJECT TREE



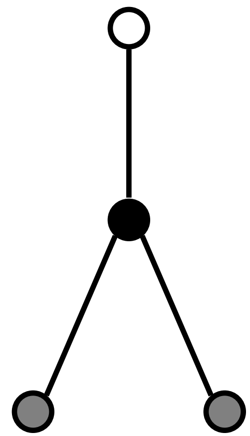
PATTERN TREES



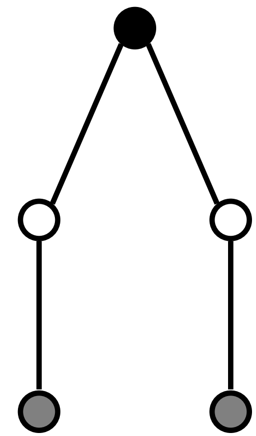
cost = 2
INV



cost = 3
NAND

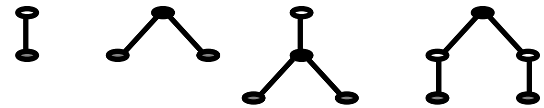


cost = 4
AND



cost = 5
OR

Example: Lib



Match of s: t1
cost = 2

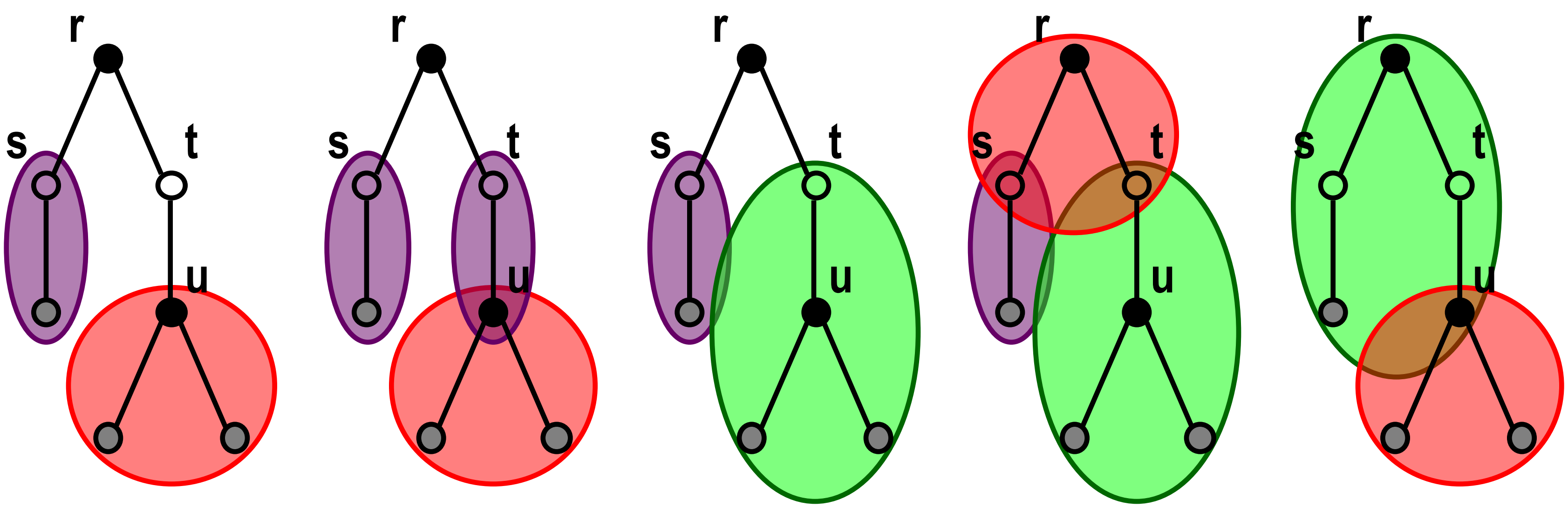
Match of u: t2
cost = 3

Match of t: t1
cost = 2+3 = 5

Match of t: t3
cost = 4

Match of r: t2
cost = 3+2+4 =9

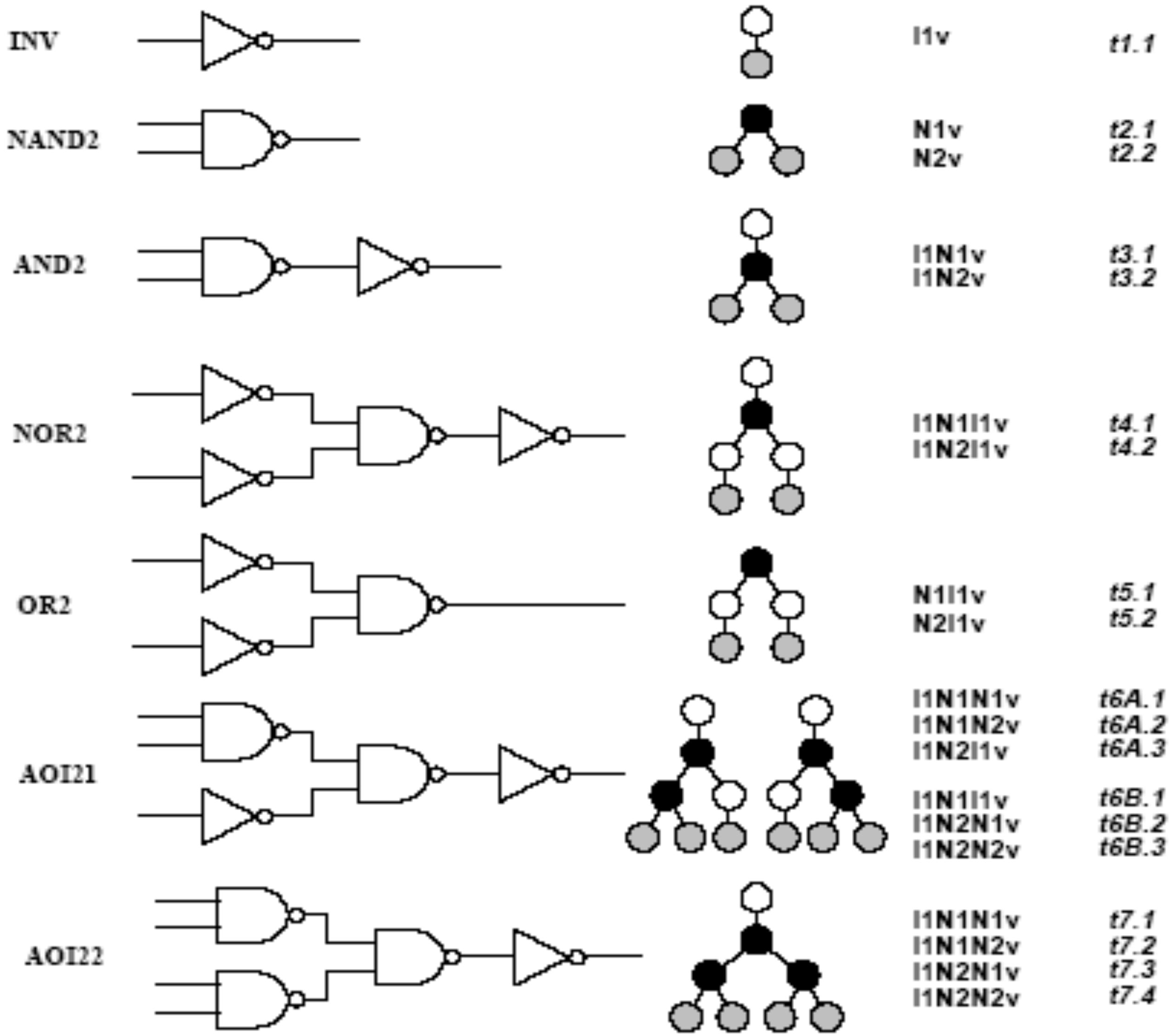
Match of r: t4
cost = 5+3 =8



Different covering problems

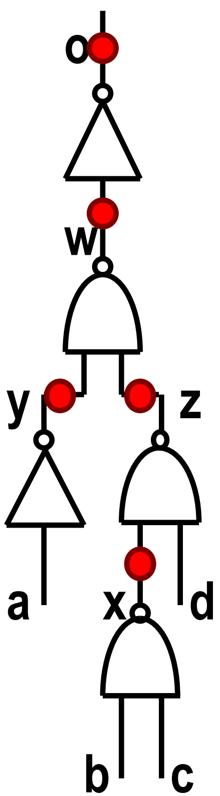
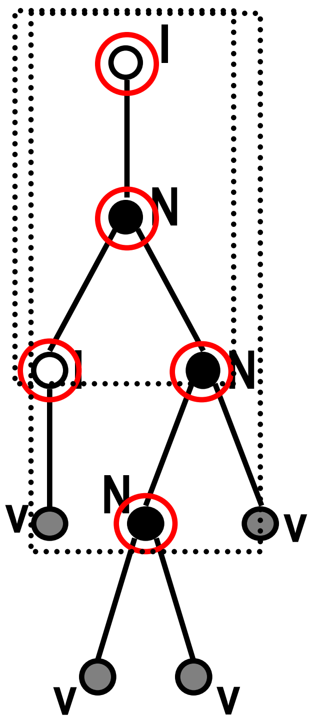
- Covering for minimum area:
  - Each cell has a fixed area cost (label)
  - Area is additive:
    - Add area of match to cost of sub-trees
- Covering for minimum delay:
  - Delay is fanout independent
    - Delay computed with (max, +) rules
    - Add delay of match to highest cost of sub-trees
  - Delay is fanout dependent
    - Look-ahead scheme is required

Simple library



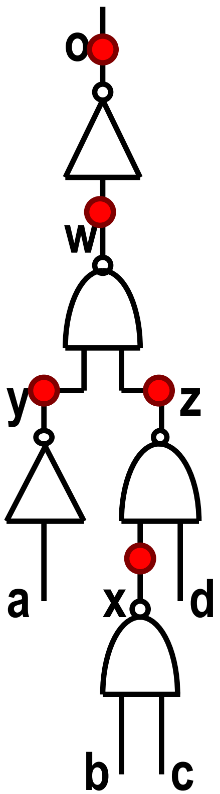
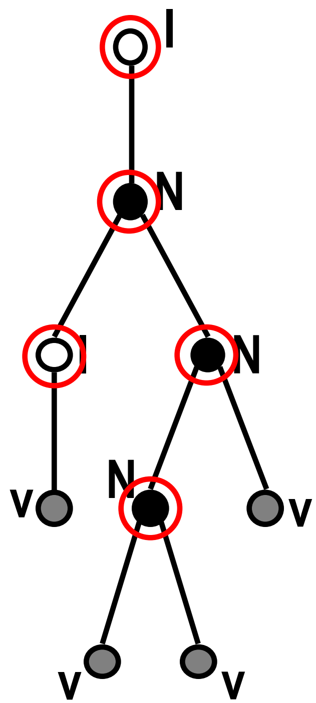
Example – minimum area cover

- **Area cost:** INV:2 NAND2:3 AND2: 4 AOI21: 6

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

Example – minimum delay cover

- Fixed delays: INV:2 NAND2:4 AND2: 5 AOI21: 10
- All inputs are stable at time 0, except for $t_d = 6$

Network	Subject graph	Vertex	Match	Gate	Cost
		x	t2	NAND2(b,c)	4
		y	t1	INV(a)	2
		z	t2	NAND2(x,d)	6+4 = 10
		w	t2	NAND2(y,z)	10 + 4 = 14
		o	t1	INV(w)	14 + 2 = 16
			t3	AND2(y,z)	10 + 5 = $\boxed{15}$
			t6B	AOI21(x,d,a)	10 + 6 = 16

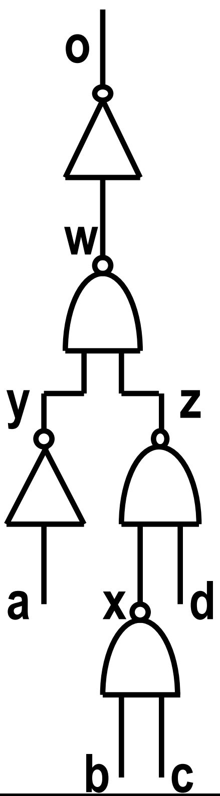
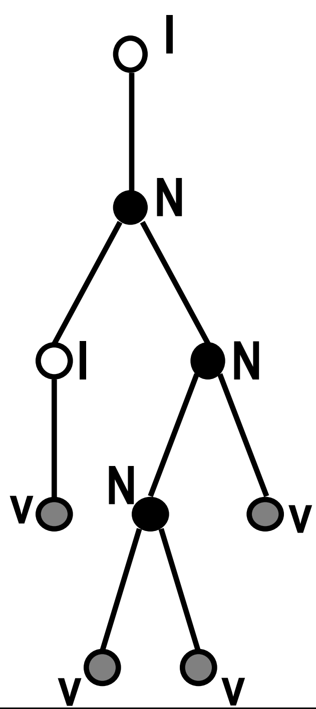
Minimum-delay cover for load-dependent delays

- **Model**
  - Gate delay is $d = \alpha + \beta$ cap_load
  - Capacitive load depends on the driven cells (fanout cone)
  - There is a finite (possibly small) set of capacitive loads
- **Algorithm**
  - Visit subject tree bottom up
  - Compute an array of solutions for each possible load
  - For each input to a matching cell, the best match for the corresponding load is selected
- **Optimality**
  - Optimum solution when all possible loads are considered
  - Heuristic: group loads into bins

Example – minimum delay cover

- Delays: INV:1+load NAND2: 3+load AND2: 4+load AOI21: 9+load
- All inputs are stable at time 0, except for $t_d = 6$
- All loads are 1

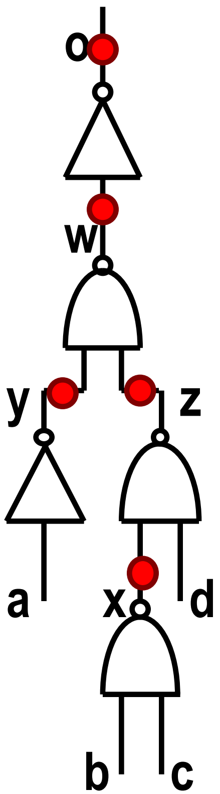
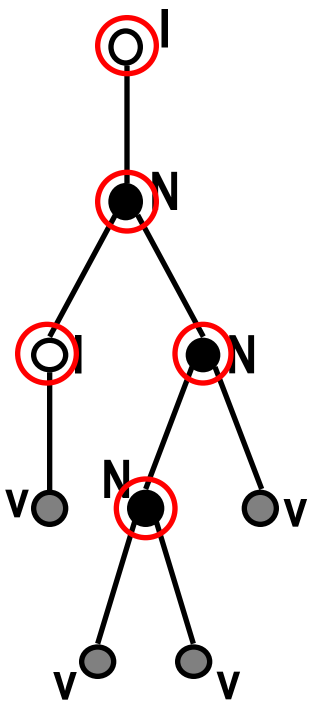
Same as before !

Network	Subject graph	Vertex	Match	Gate	Cost
		x	t2	NAND2(b,c)	4
		y	t1	INV(a)	2
		z	t2	NAND2(x,d)	6+4 = 10
		w	t2	NAND2(y,z)	10 + 4 = 14
		o	t1	INV(w)	14 + 2 = 16
			t3	AND2(y,z)	10 + 5 = $\boxed{15}$
			t6B	AOI21(x,d,a)	10 + 6 = 16

Example – minimum delay cover

- Delays: INV: 1+load NAND2: 3+load AND2: 4+load AOI21: 9+load
- All inputs are stable at time 0, except for $t_d = 6$
- All loads are 1 (for cells seen so far)
- Add new cell SINV with delay 1 + ½ load and load 2
- The sub-network drives a load of 5

Example – minimum delay cover

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)			$\boxed{18.5}$

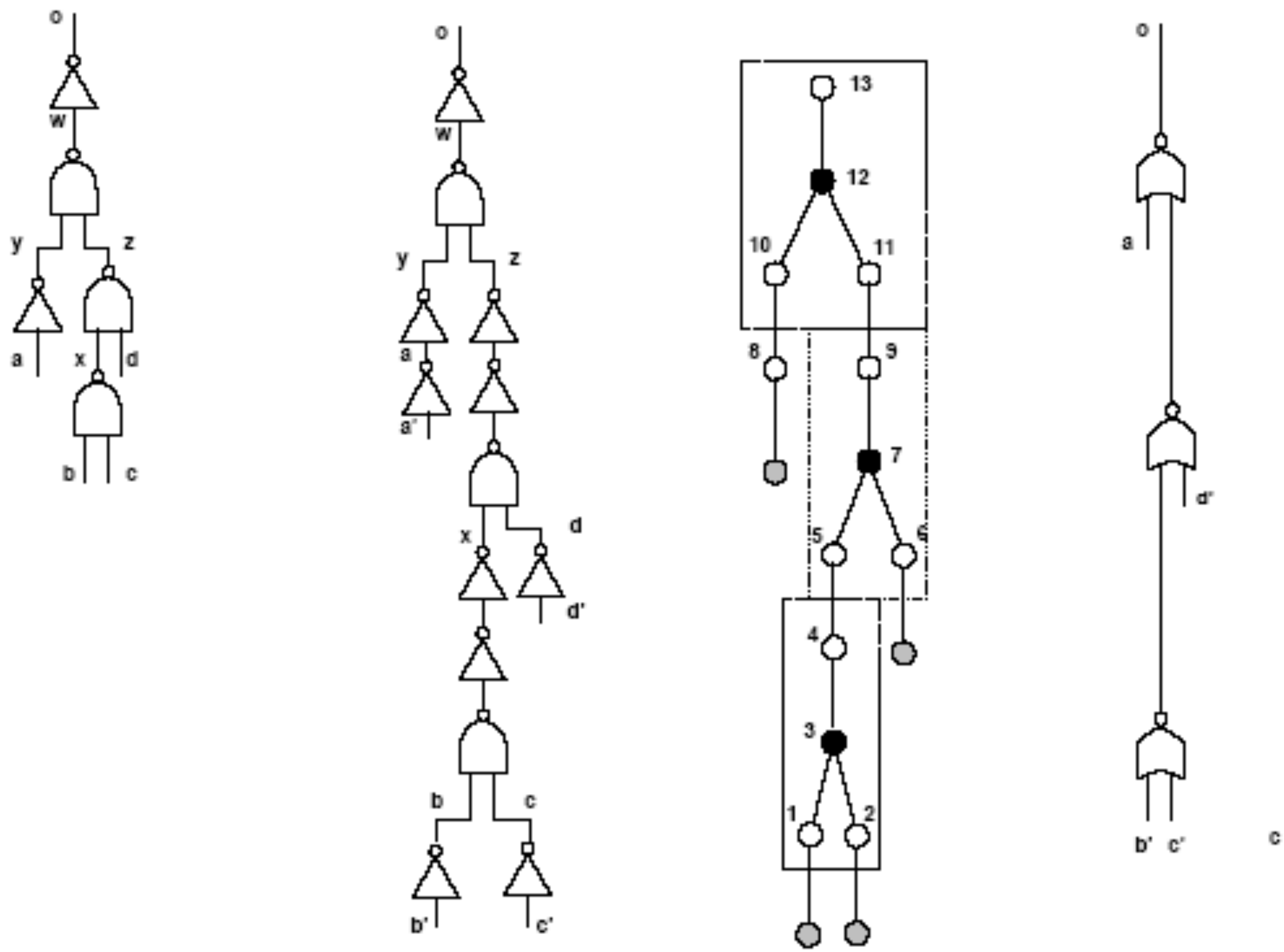
Library binding and polarity assignment

- Search for lower cost solutions allowing signals to be generated with inverted polarity
  - More cells to choose from
- Polarities can be adjusted at register and/or I/O boundaries
- Within structural covering:
  - Polarity assignment is handled by a smart trick
- Within structural covering
  - Polarity assignment is built into the formulation

Structural covering and polarity assignment

- Assume subject network is decomposed into base functions such as NAND2 and INV
- Pre-process subject network by adding inverter pairs between NANDs
- Provide I/Os with both polarity
- Add to library inverter-pair cell 2INV to the library:
  - Cell corresponds to a connection and has 0 cost
  - Unnecessary 2INV will be removed by the algorithm
- Apply bottom-up dynamic programming cover algorithm

Example



General issues with covering

- Decomposition and covering yield solutions that are locally sub-optimal at multiple fanout points
  - Covering in various logic cones is independent
  - High capacitive loads skews the solution
- Approaches
  - Implicit decomposition methods
    - Compute and map all possible decompositions of a network
  - Cell cloning and buffering at cone vertices
  - Wavefront mapping