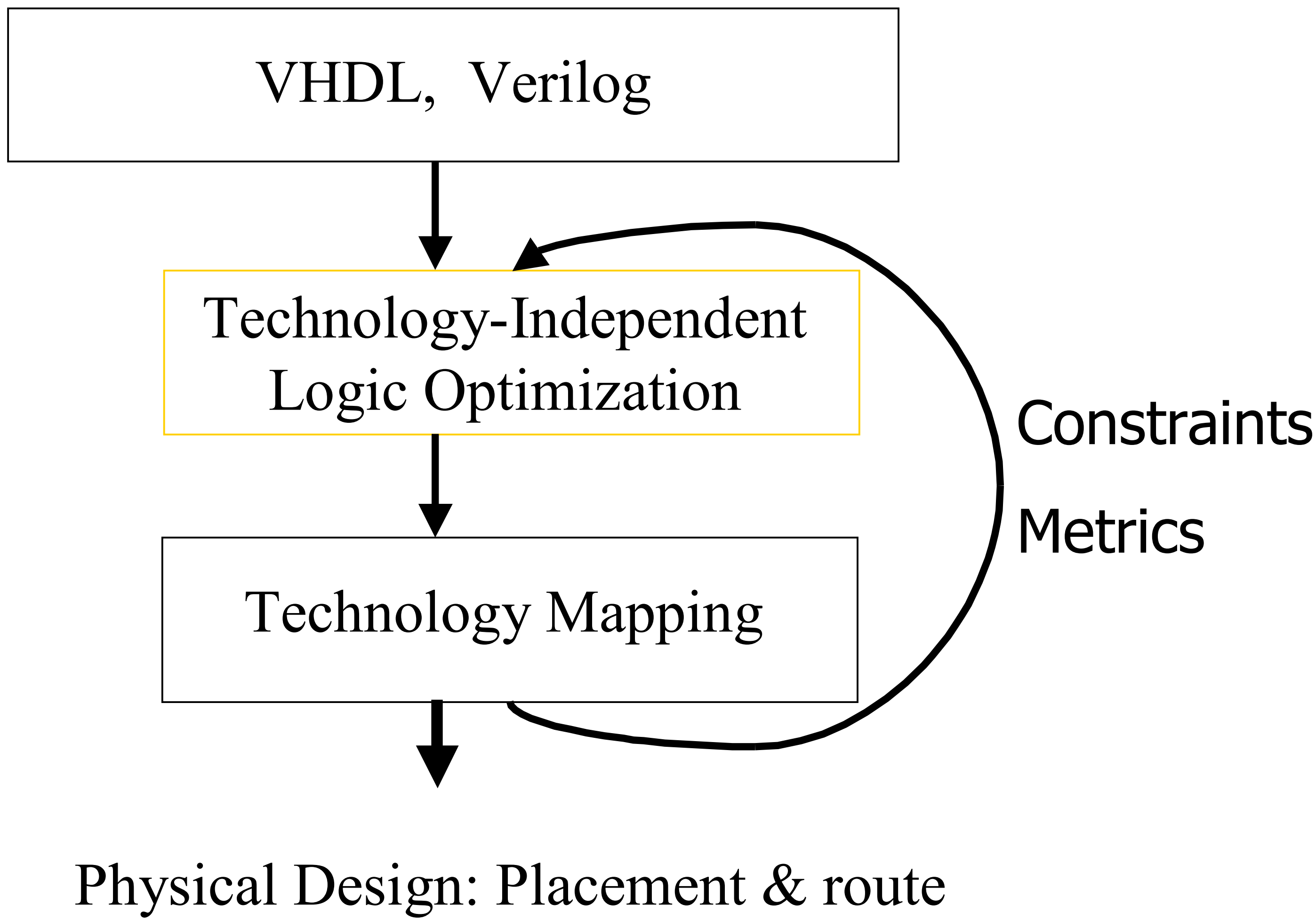
BDD-Based Logic Decomposition

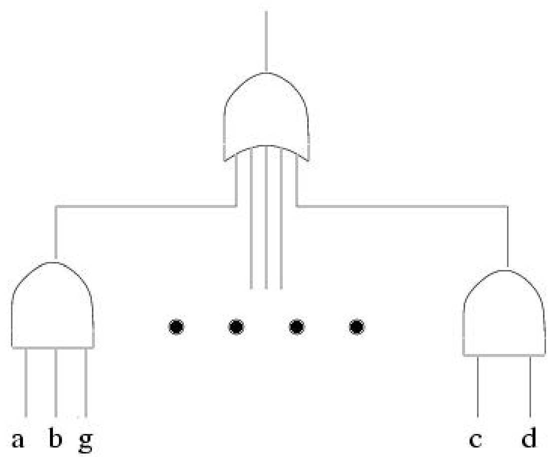
Sources:
C. Yang, M. Ciesielski
F. Brewer, J. Roy, I. Markov

Logic Synthesis Flow

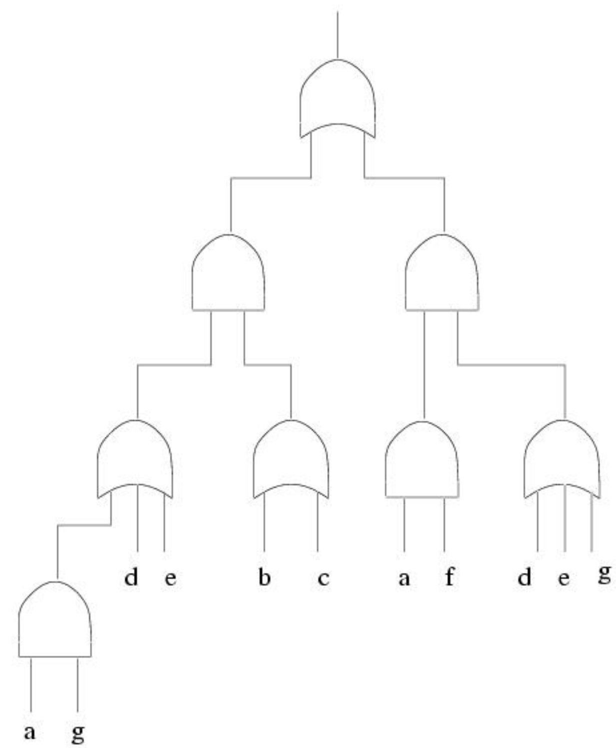


Multi-Level Logic Synthesis

abg + acg + adf + aef + afg + bd + ce + be + cd



(b+c)(d+e+ag) + (d+e+g)af



Minimizing # of product terms

Quine-McCluskey method (minterm)

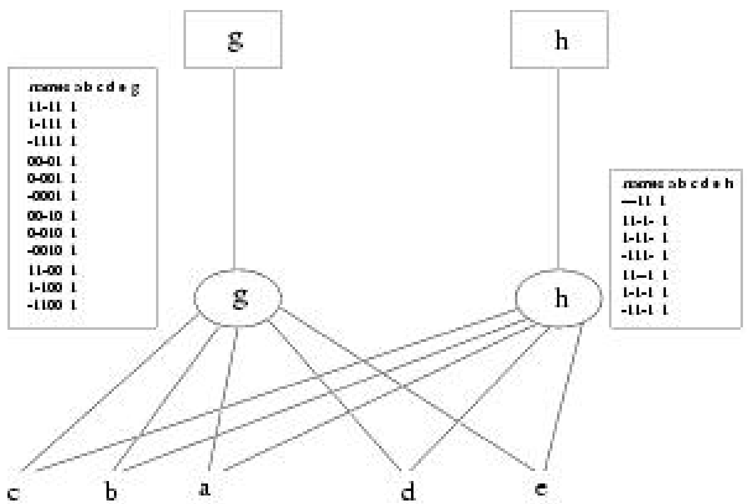
Espresso (SOP)

Minimizing # literals, gates, delay.

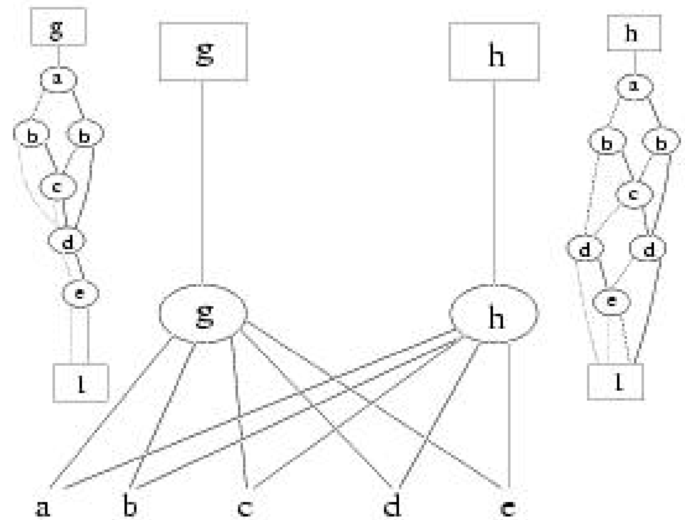
Algebraic factorization.

SIS (factored form)

Global Representation of Boolean Network

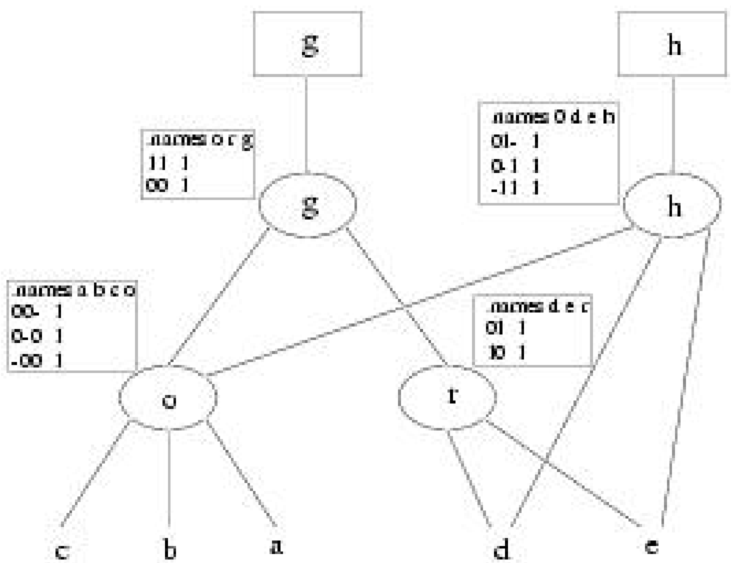


(i) Two-level SOP representation

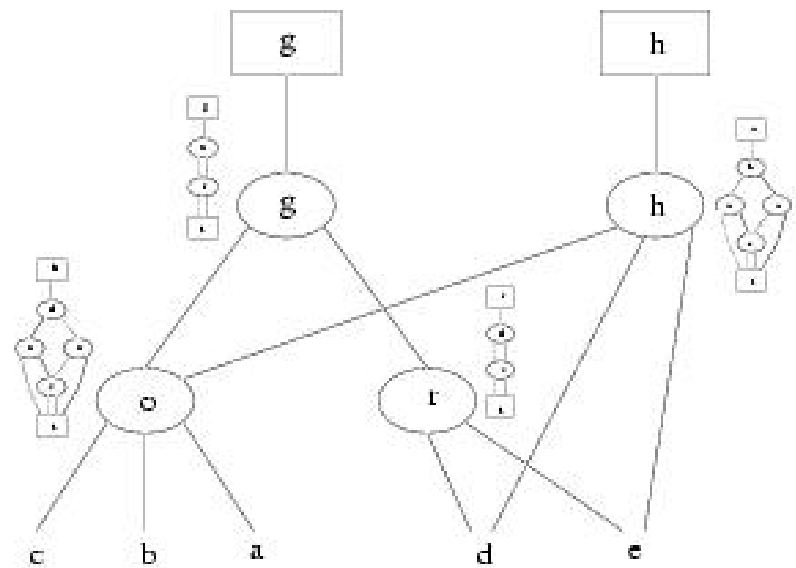


(j) Global BDD representation

Boolean Network



(g) Multi-level SOP representation



(h) Local BDD representation

Boolean Decomposition – previous work

- Ashenhurst [1959], Curtis [1962]
  - Tabular method based on cut: bound/free variables
  - BDD implementation:
    - Lai et al. [1993, 1996], Chang et al. [1996]
    - Stanion et al. [1995]
- Roth, Karp [1962]
  - Similar to Ashenhurst, but using cubes, covers
  - Also used by SIS
- Factorization based
  - SIS, algebraic factorization using cube notation
  - Bertacco et al. [1997], BDD-based recursive bidecomp.

Functional Decomposition

- Decompose Network into Composed Functions
  - Minimize Depth and Redundancy

$$f(x_1, x_2..x_n) = g(h(x_1, x_2..x_m), x_{m+1}..x_n)$$

- Decomposition is disjoint if support of g and h are.
- Disjoint decomposition exists iff each

$$f\Big|_{x_{m+1}..x_n}(x_1, x_2..x_m) = f_1(x_1, x_2..x_m) \vee f_2(x_1, x_2..x_m)$$

  - I.e. high degree of internal symmetry
- Non-disjoint decompositions are more common

Ashenhurst Decomposition

Goal: $f(X) = F(\Phi(Y), Z)$.

Example: $F = w'x'z' + wx'z + w'yz + wyz' = f(\Phi(w,z), x, y) = \Phi x' + \Phi' y$.

xy \ wz	0	1	2	3
0	1	0	0	1
1	1	1	1	1
2	0	0	0	0
3	0	1	1	0

(a) $\mu = 2$

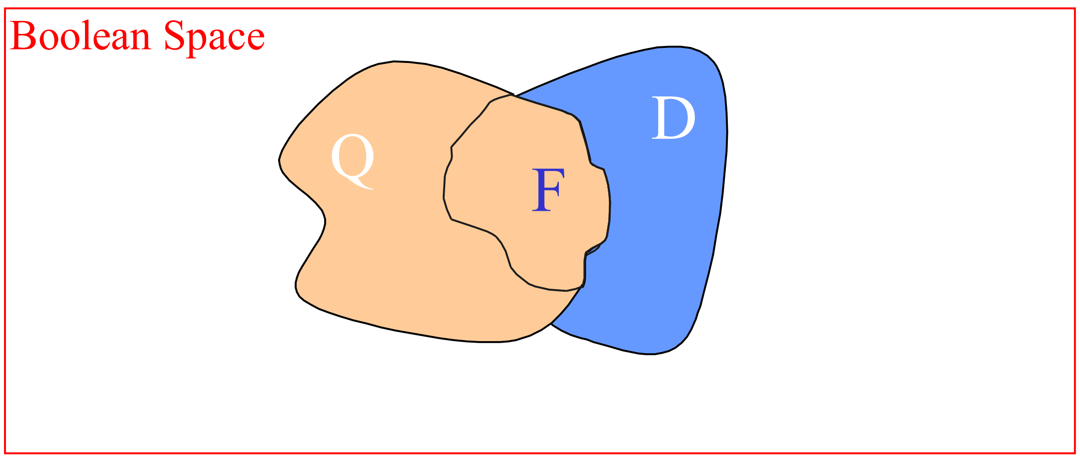
xw \ yz	0	1	2	3
0	1	0	1	1
1	0	1	1	1
2	0	0	0	1
3	0	0	1	0

(b) $\mu > 2$

u is the number of distinct columns

Boolean AND Decomposition

Goal : find D, Q such that $F = Q * D$.



$F = e + bd;\ \ Q = e + b;\ \ \ D = e + d$

Boolean Division

- G is a Boolean divisor of F if there exist H and R such that

$$F = G\,H + R, \text{ and } G\,H \neq 0.$$

- G is said to be a factor of F if, in addition, R=0, that is:

$$F = G\,H.$$

  where H is the quotient, R is the remainder.

  Note: H and R are not unique.

Many Boolean Factors!

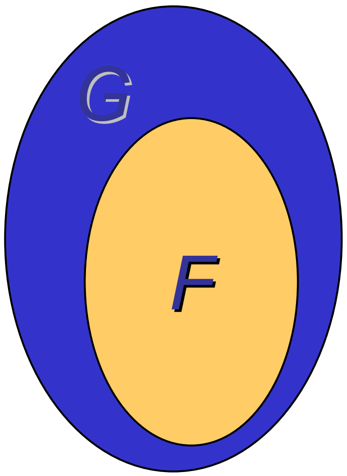
G is a Boolean factor of F iff $F \subseteq G$

(i.e. $FG' = 0$, or $G' \subseteq F'$).

Example:

$F = a + bc; \quad G = (a+c)$

$F = (a+b)(a+c); \; R = 0$



Proof:

$\Rightarrow$: G is a Boolean factor of F. Then $\exists H$ s.t. $F = GH$;
Hence, $F \subseteq G$ (as well as $F \subseteq H$).

$\Leftarrow$: $F \subseteq G \Rightarrow F = GF = G(F + R) = GH$.
(Here R is any function $R \subseteq G'$)

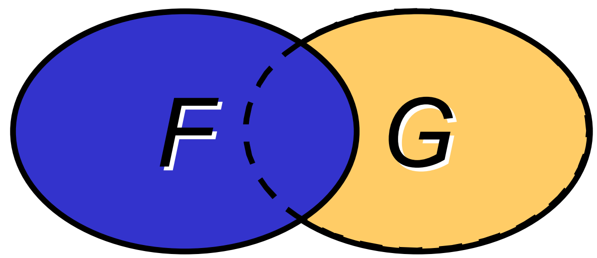
More Boolean Divisors

G is a Boolean divisor of F if and only if $F\,G \neq 0$.

Example:

$F = ab+ac+bc;\ G = b+c$

$F= a(b+c)+bc;\ R=bc$



Proof:

$\Rightarrow$: $F = GH + R,\ GH \neq 0 \Rightarrow FG = GH + GR$. Since $GH \neq 0,\ FG \neq 0$.

$\Leftarrow$: Assume that $FG \neq 0$. $F = FG + FG' = G(F + K) + FG'$. (Here $K \subseteq G'$.)
Then $F = GH + R$, with $H = F + K,\ R = FG'$. Since $GH = FG \neq 0$, then $GH \neq 0$.

Algebraic Factorization

- <u>Objective</u> : Obtain a minimized factored form
- x and x′ treated as different variables
- Boolean algebra rules: $x + x' = 1$; $x\,x' = 0$, not applied.
- A Boolean formula treated as polynomial formula
- Two-level techniques for Boolean simplification.
- Polynomial representation is the bottleneck in Boolean factorization

Boolean Division

- F = abg + acg + adf + aef + afg + bd + ce + be + cd. (23 lits)
- Algebraic : F = (b + c)(d + e + ag) + (d + e +g)af. (11 lits)
- Boolean: F = (af + b + c)(ag + d + e). (8 lits)

Algebraic Logic Synthesis Issues

- Weak Boolean factorization
- Difficult to perform XOR and MUX decomposition.
- Separate platforms for Boolean operations and factorization.

- Goal: use BDD platform for both Boolean operations and Factorization
  - BDD structure used to find good factors
  - Maybe form *non*-conjunctive decompositions

BDD – Synthesis

- BDD is canonical
  - Variable reordering implicit logic simplification
  - Some redundancy removed
- BDD is efficient factored form
  - Reduced representation should lead to efficient matching

- Problem : How to translate a BDD into a multi-level (circuit) representation?

Shannon Expansion

A Boolean function $f(x_1, x_2, \cdots, x_i, \cdots, x_n)$ can be expressed as,

$$\begin{aligned} f(\cdots, x_i, \cdots) &= x_i f(\cdots, 1, \cdots) + x_i' f(\cdots, 0, \cdots) \\ &= x_i f_i^1 + x_i' f_i^0; \end{aligned}$$

$$\begin{aligned} f(\cdots, x_i, \cdots) &= (x_i + f(\cdots, 0, \cdots))(x_i' + f(\cdots, 1, \cdots)) \\ &= (x_i + f_i^0)(x_i' + f_i^1); \end{aligned}$$

f_i^1 : *positive cofactor;* f_i^0 : *negative cofactor*

Orthogonal Expansion

- Shannon Expansion is just a special case of linear orthonormal expansion:

Given set of functions: $g_i(x_1..x_n)$

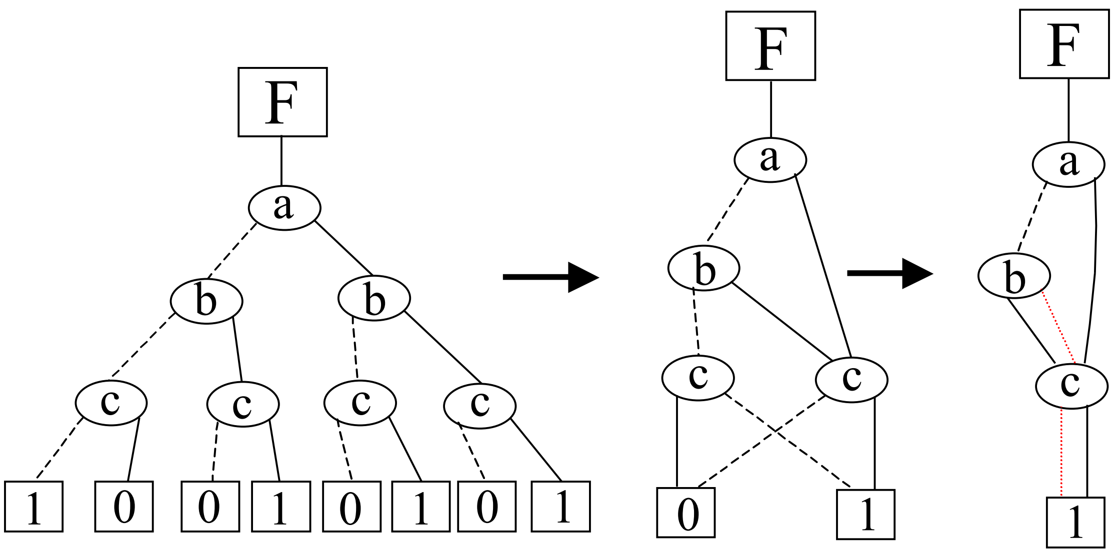
Where $\bigcup_i g_i = T$ and $\forall_{i,j,i \neq j} g_i g_j = 0$

$$f(x_1..x_n) = \bigcup_i g_i(x_1..x_n) f\big|_g$$

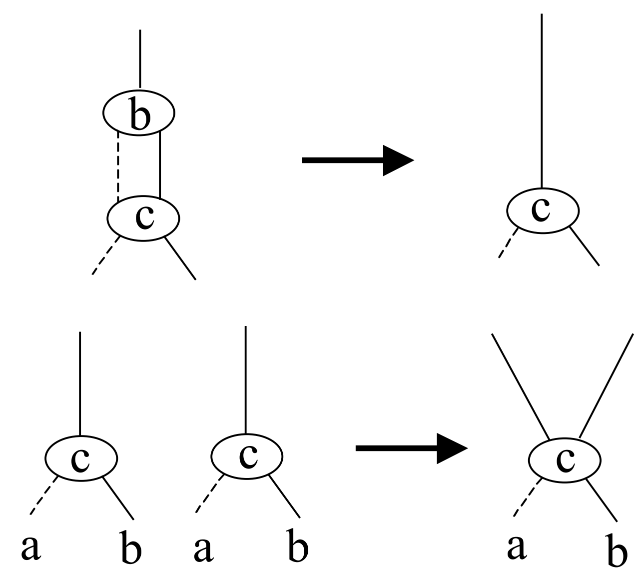
$f\big|_g (x_1..x_n)$ is the generalized co-factor of f by g.

BDD Reduction

Example: $F = ac + bc + a'b'c'$;

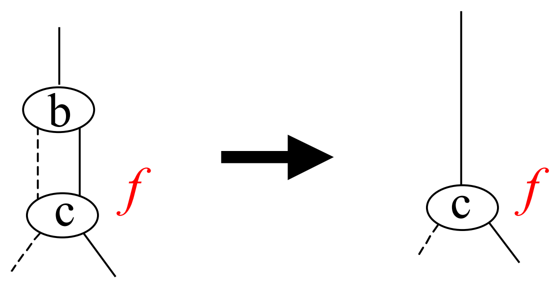


Reduction Rules :

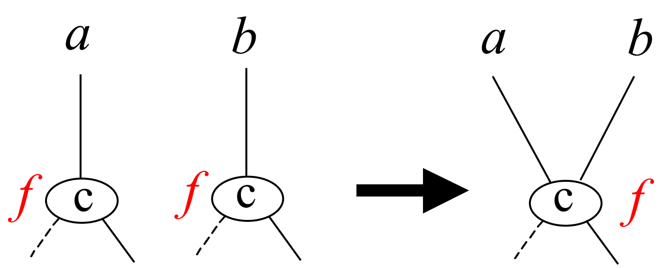


BDD -- Synthesis

- A BDD is an implicitly factored representation

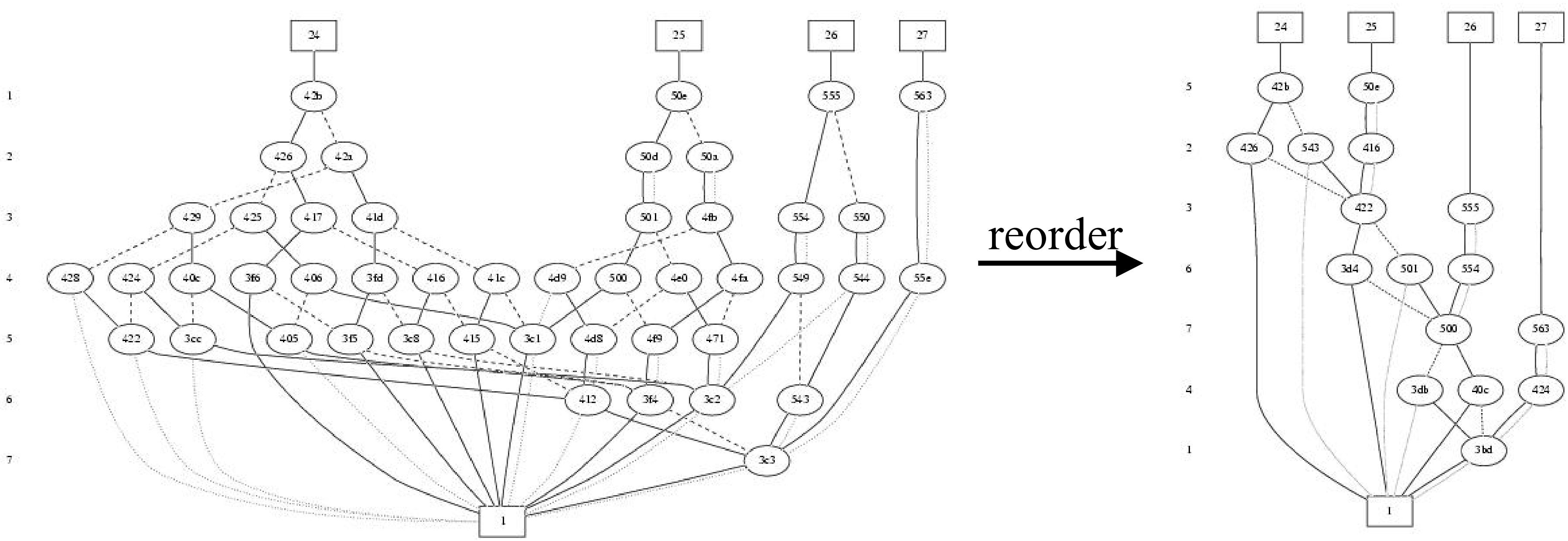


$$b' f + b f = f$$



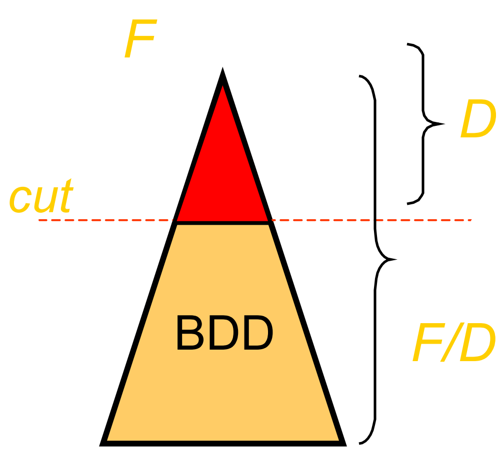
$$a f + b f = (a + b) f$$

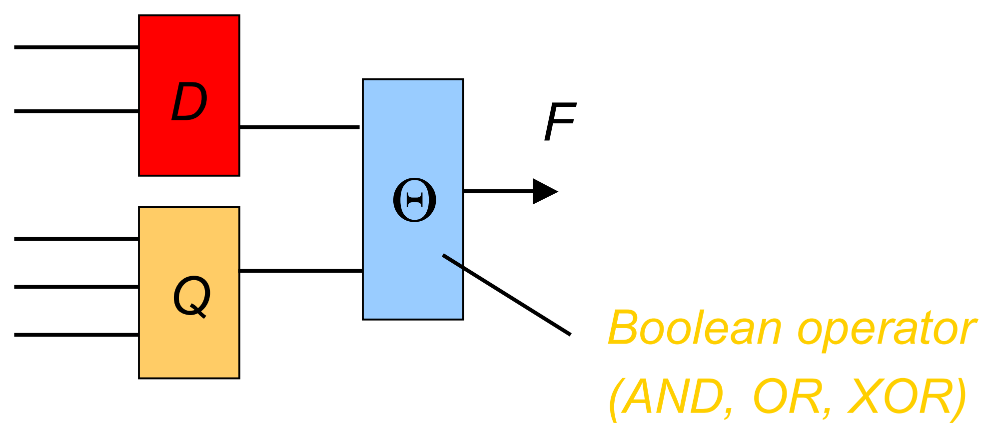
Variable Ordering



Non-Disjoint Decomposition

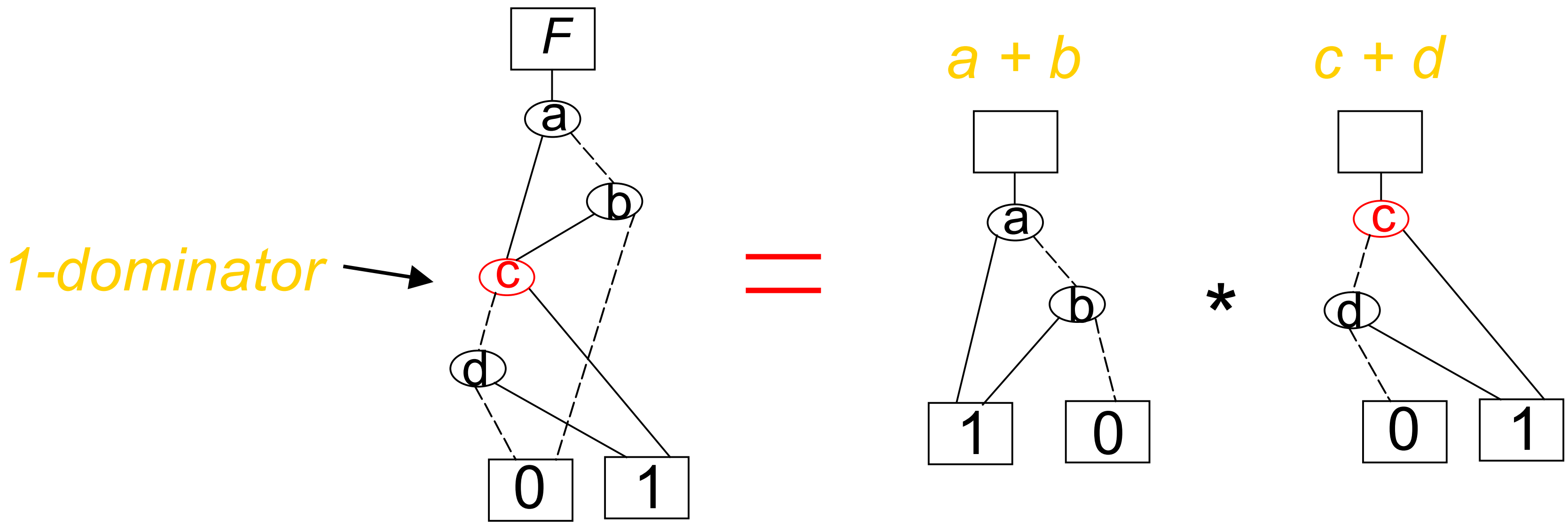
- Bi-decomposition: F = D Θ Q
  - uses BDD cut
    - *Not* a partition of input support variables!
  - The top set defines a divisor D
  - The bottom set defines the quotient Q (sort of …)





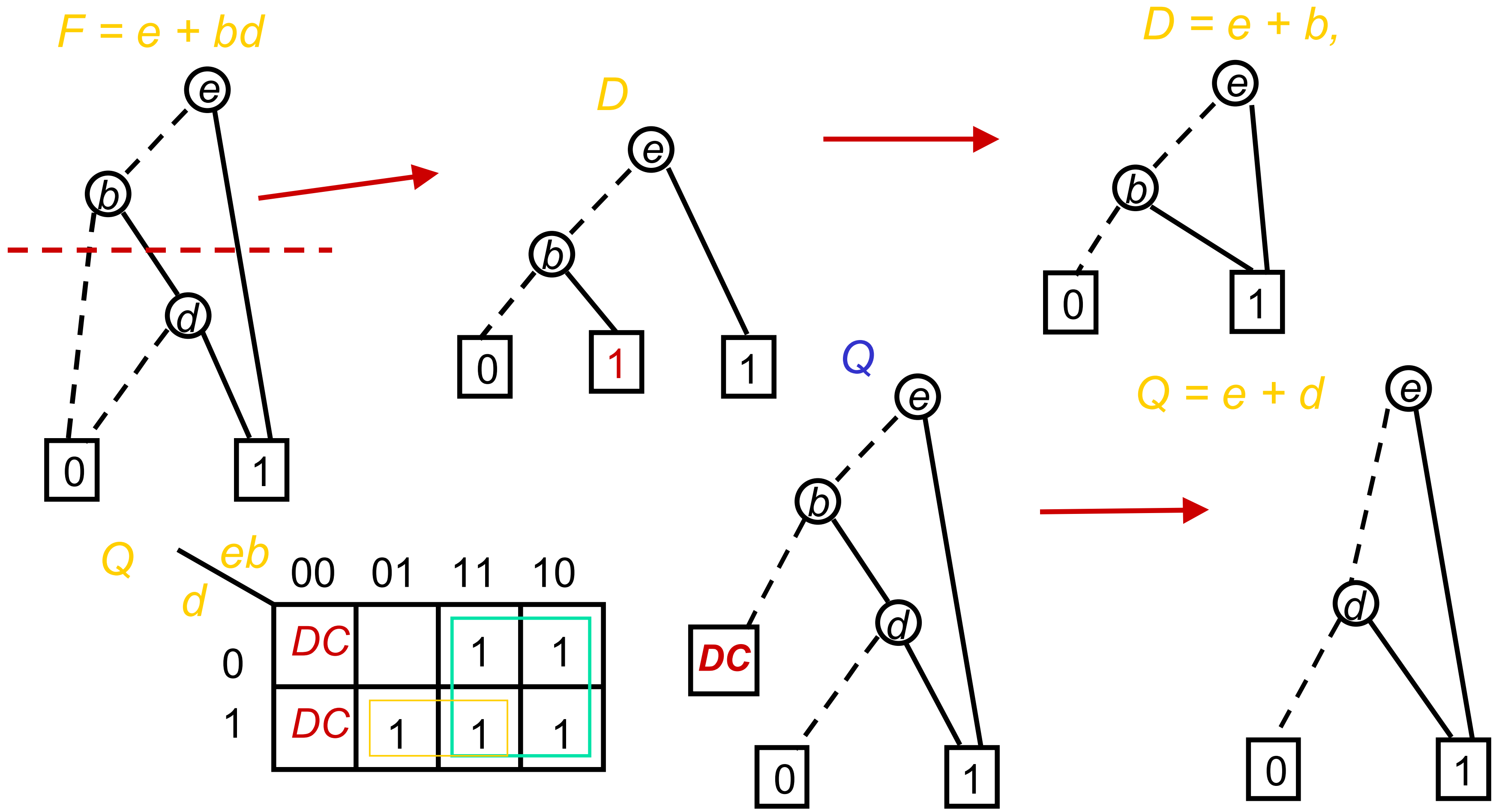
1-dominator: Karplus [1988]:

- *1-dominator* is a node that belongs to every path from root to 1-terminal.



- *1-dominator* defines *algebraic conjunctive (AND)* decomposition: *F = (a+b)(c+d)*.

Eg. AND Decomposition: F = D Q

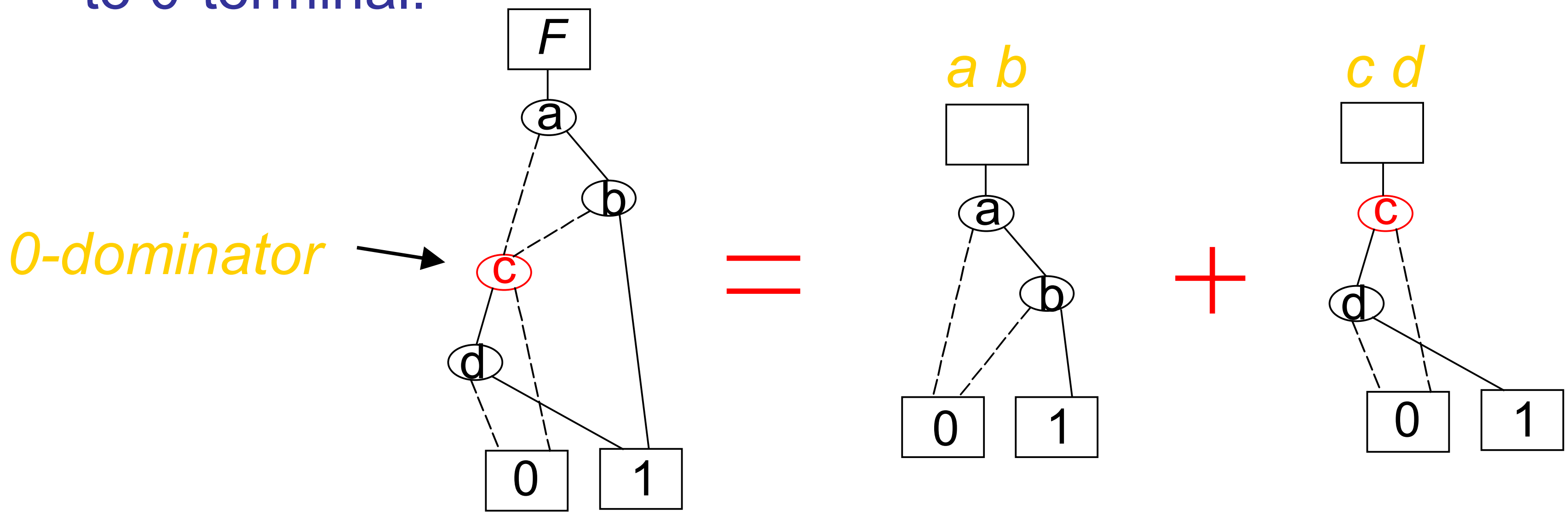


Q

d \ eb	00	01	11	10
0	DC		1	1
1	DC	1	1	1

0-dominator

0-dominator is a node belonging to every path from root to 0-terminal.



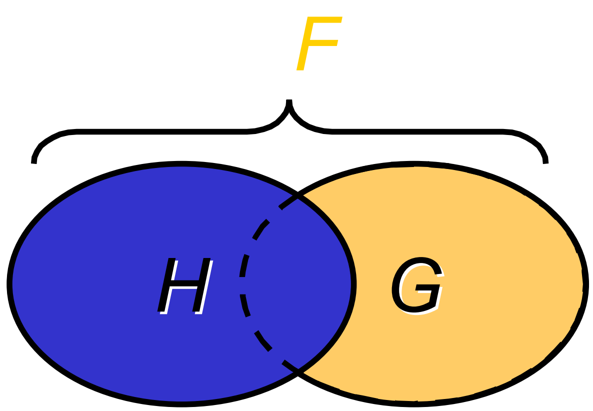
- *0-dominator* defines *algebraic disjunctive (OR)* decomposition: $F = ab + cd$.

Disjunctive Decomposition

- Disjunctive (OR) decomposition: $F = G + H$.

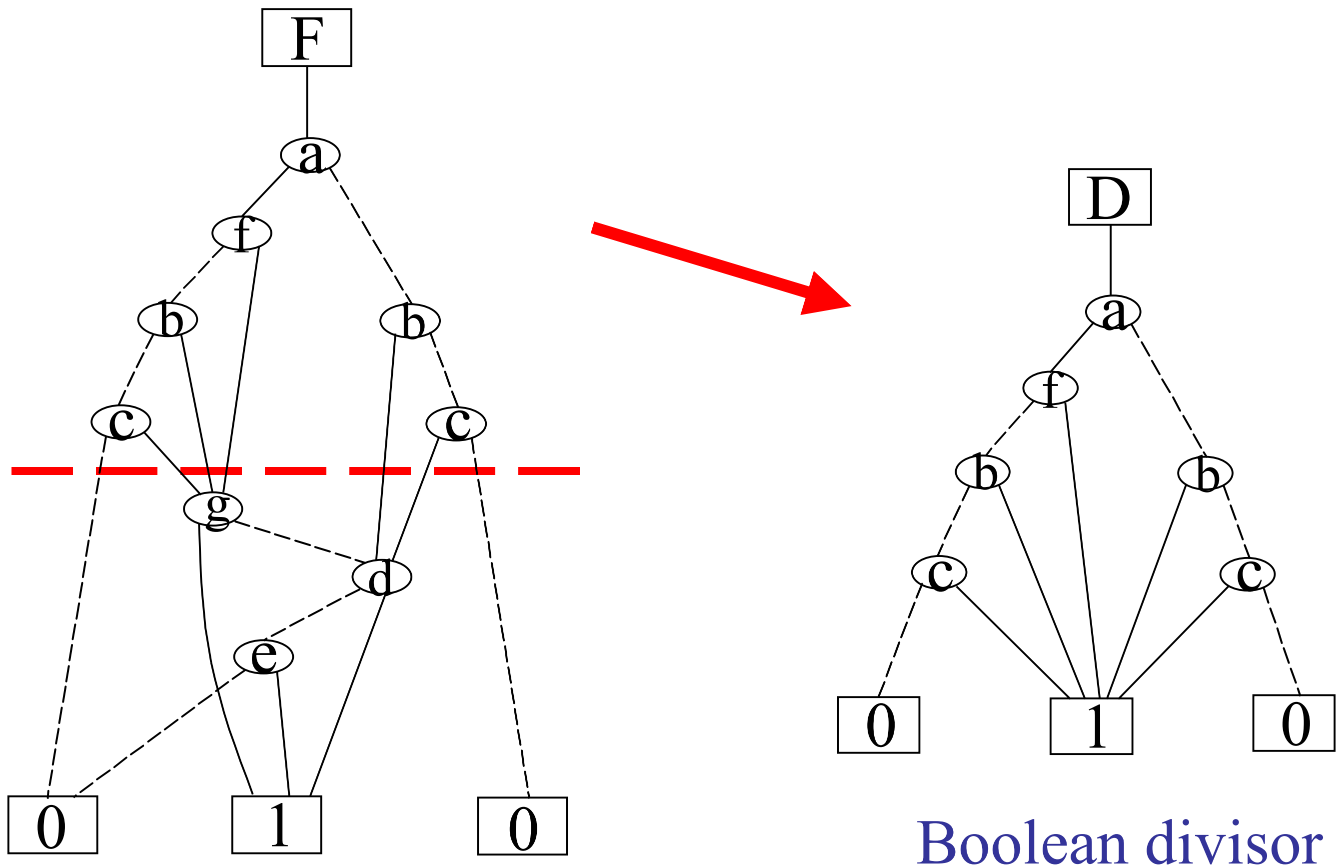
Boolean function F has disjunctive decomposition iff $F \supseteq G$. For a given G, H must satisfy:

$F' \subseteq H' \subseteq F' + G$.



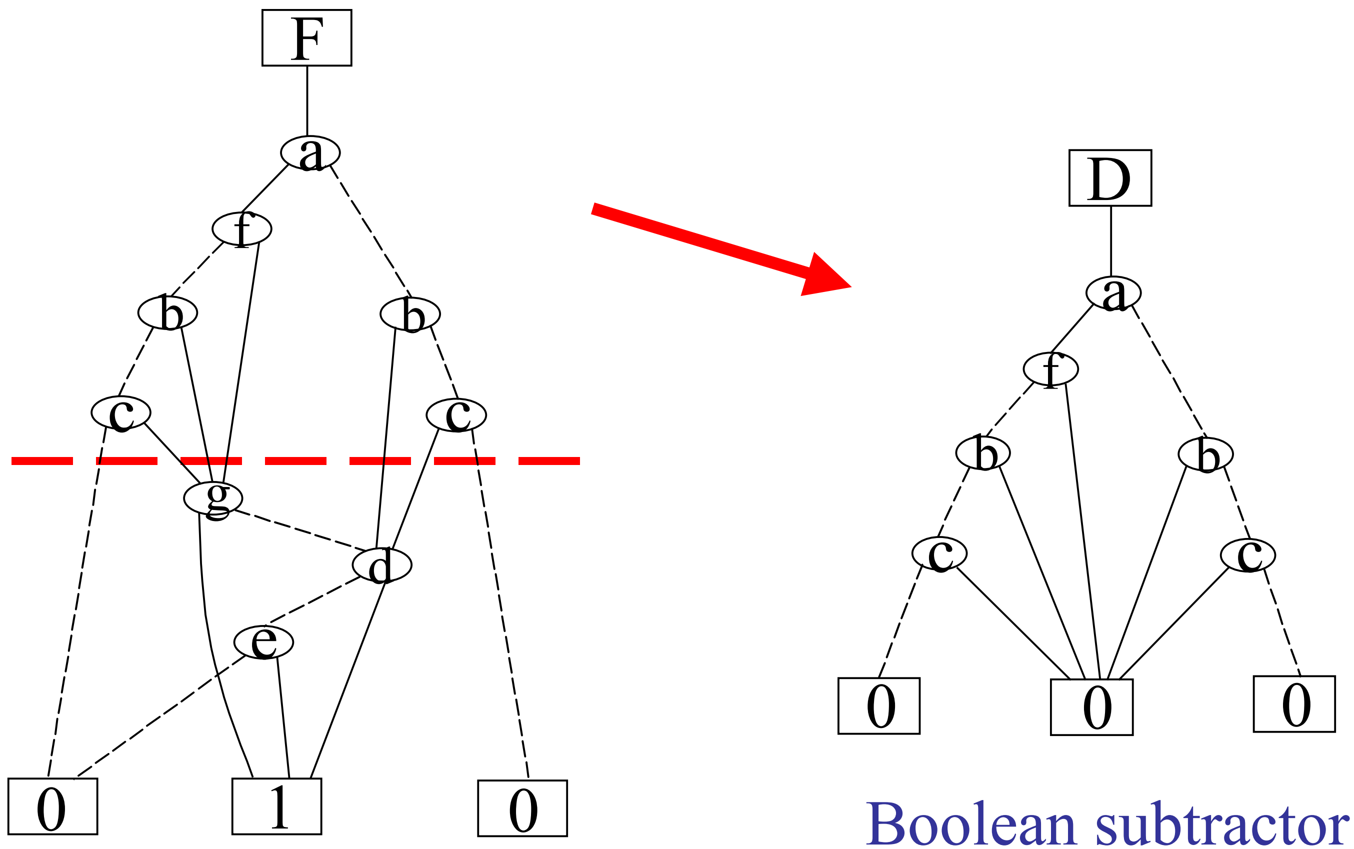
- For a given (F,G), this provides a recipe for H .

Generalized Dominator (And)



Boolean divisor

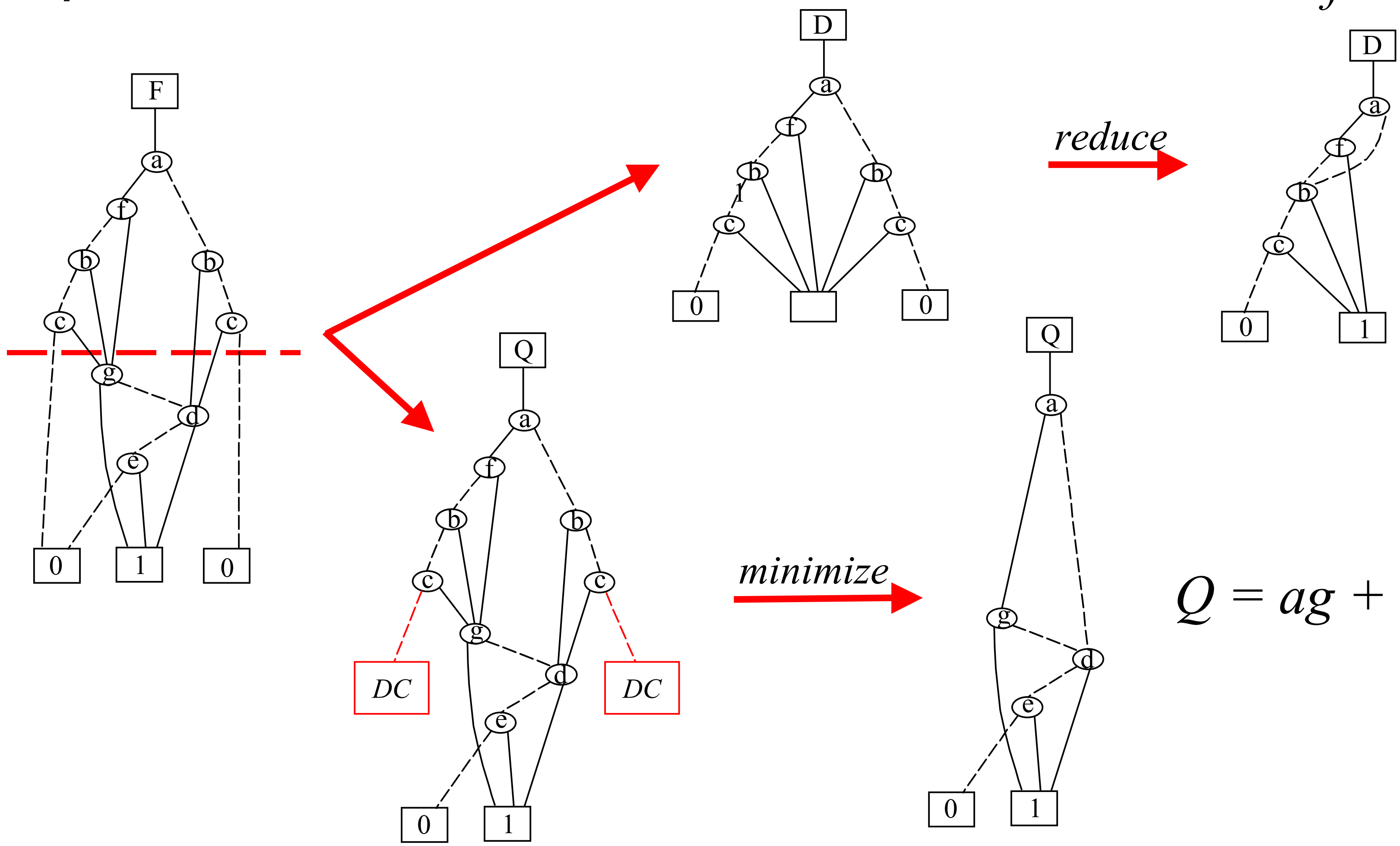
Generalized Dominator (or)



Boolean subtractor

Boolean Division via Dominator

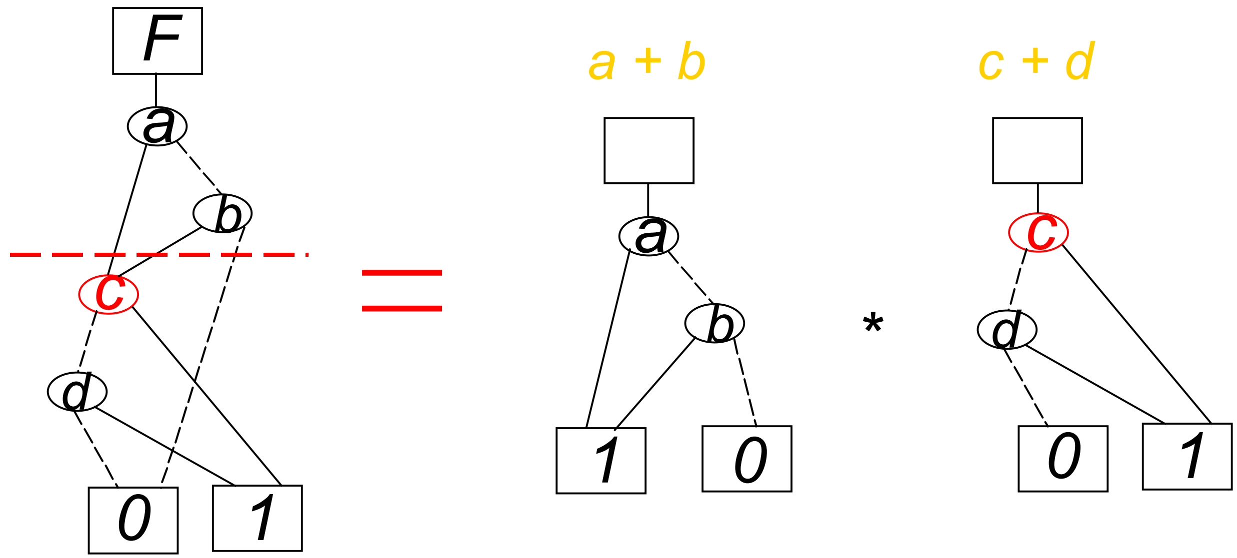
$$D = af + b + c$$



$$Q = ag + d + e$$

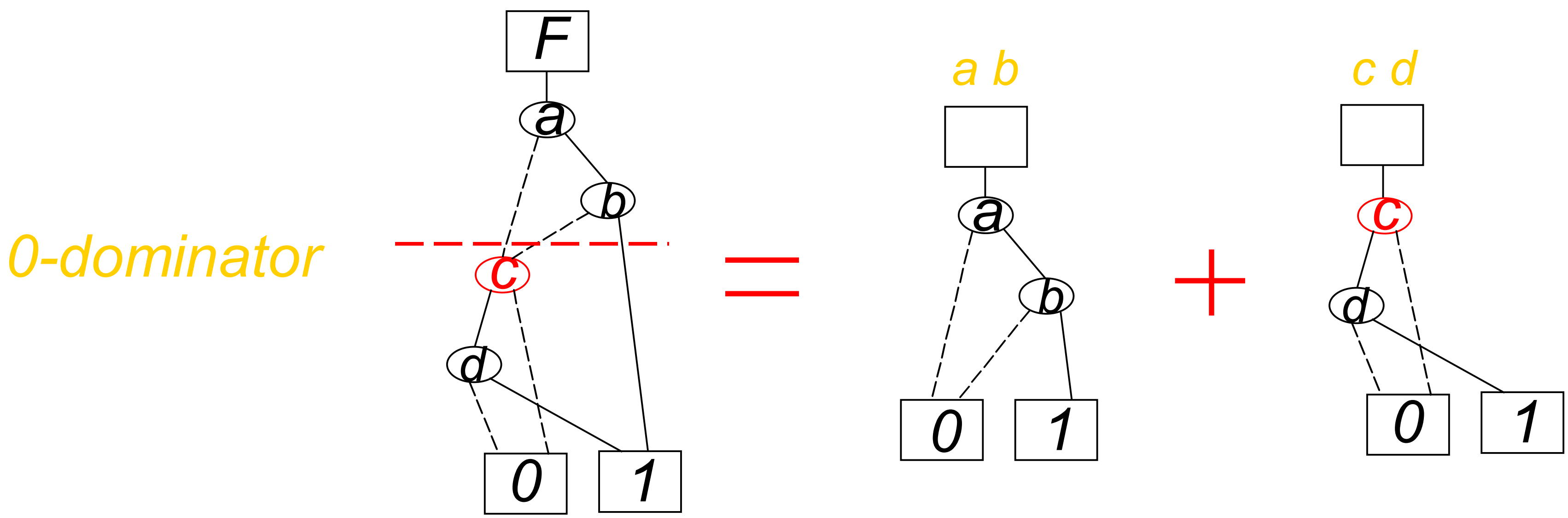
Special Case: 1-dominator

1-dominator



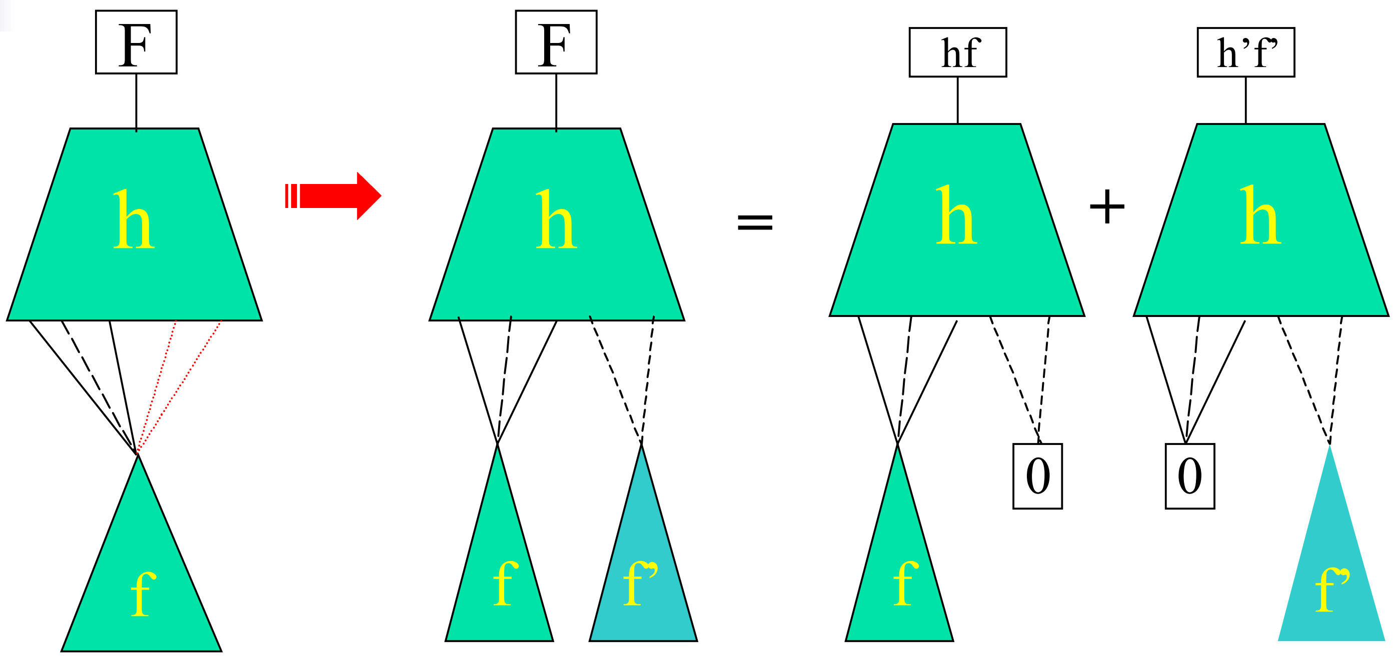
F = (a+b)(c+d)

Special Case: 0-dominator



F = ab + cd

Algebraic XOR Decomposition

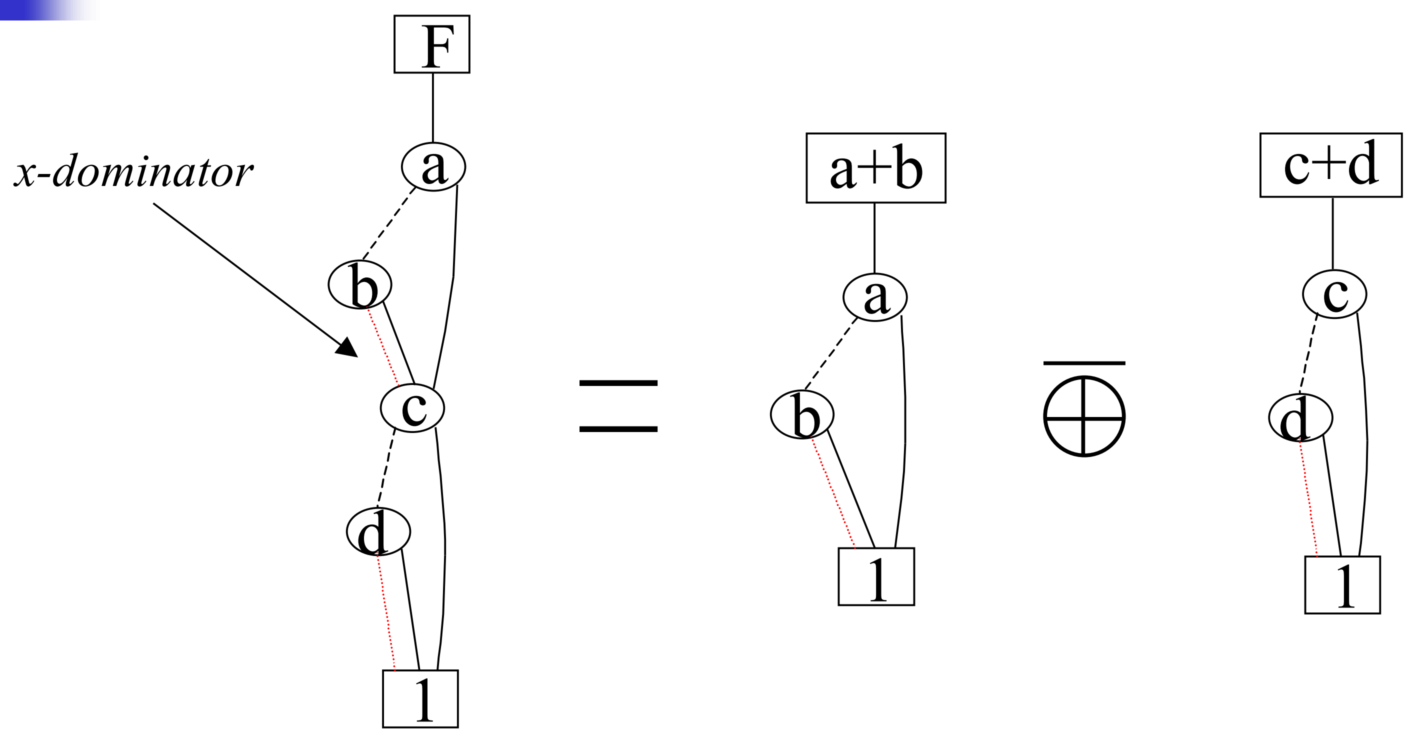


= complement edge

= 1-edge edge

= 0-edge edge

Algebraic XOR Decomposition: x-dominator

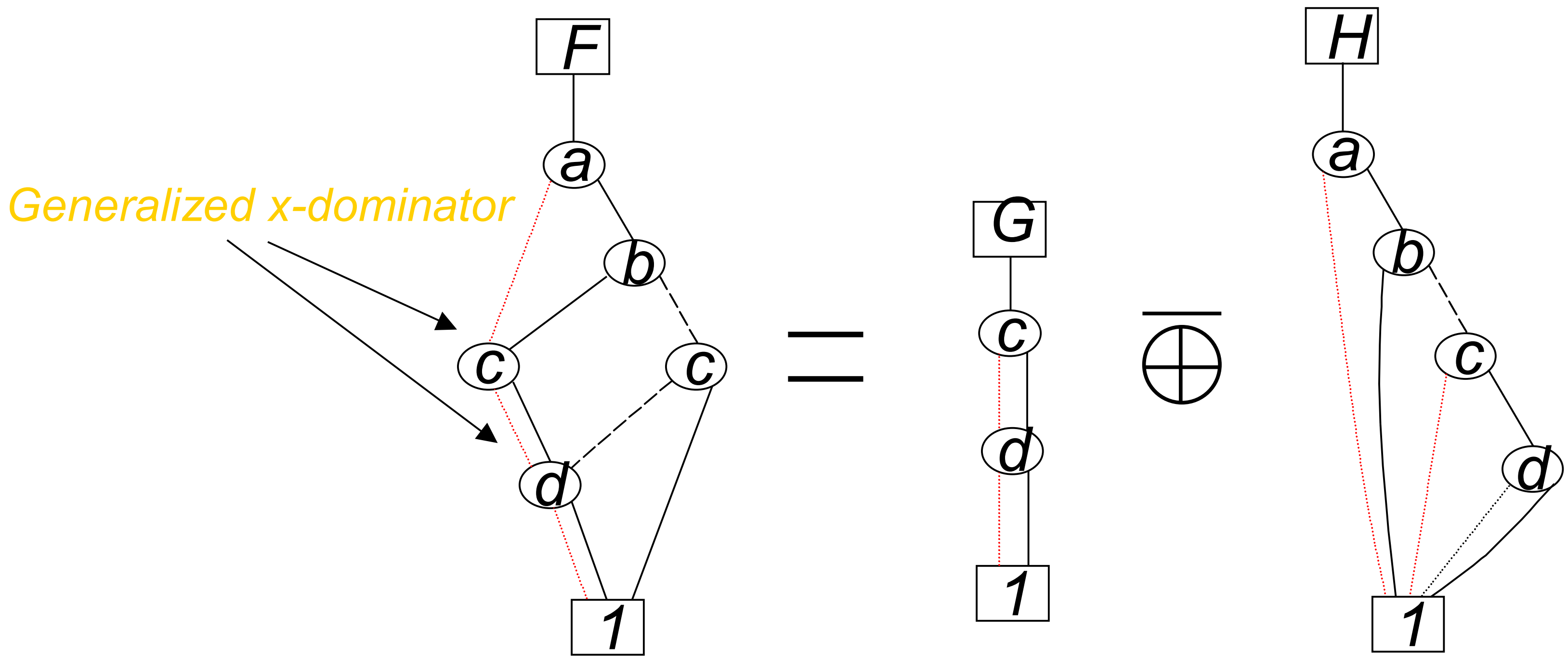


- X-dominator nodes contain *every* path in BDD
  - Denotes equivalence of upper and lower parts
  - Complement Edges are the key to finding such nodes

Boolean XOR Decomposition: Generalized x-dominators

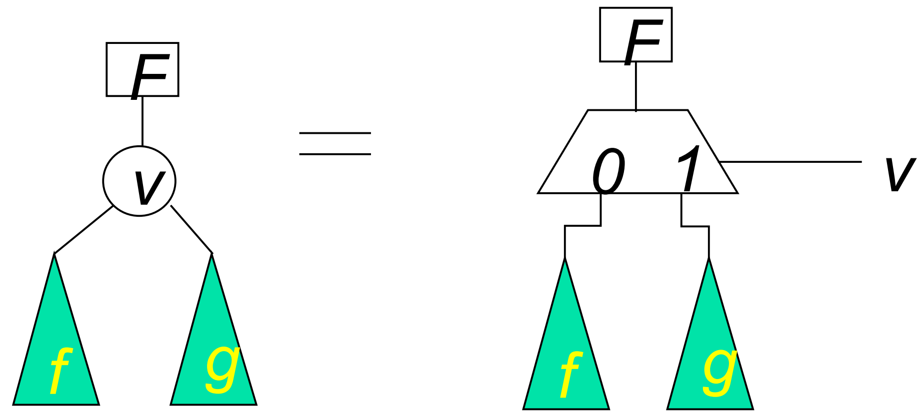
Generalized X-dom: any node with incoming true and complement edges.

Given F and G, there exists H : F = G ⊗ H; H = F ⊗ G.

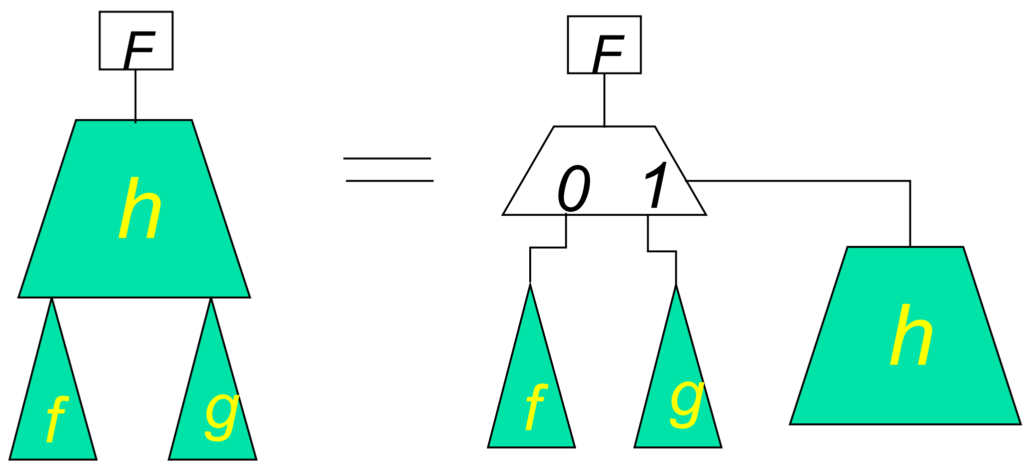


MUX Decomposition

- Simple MUX decomposition

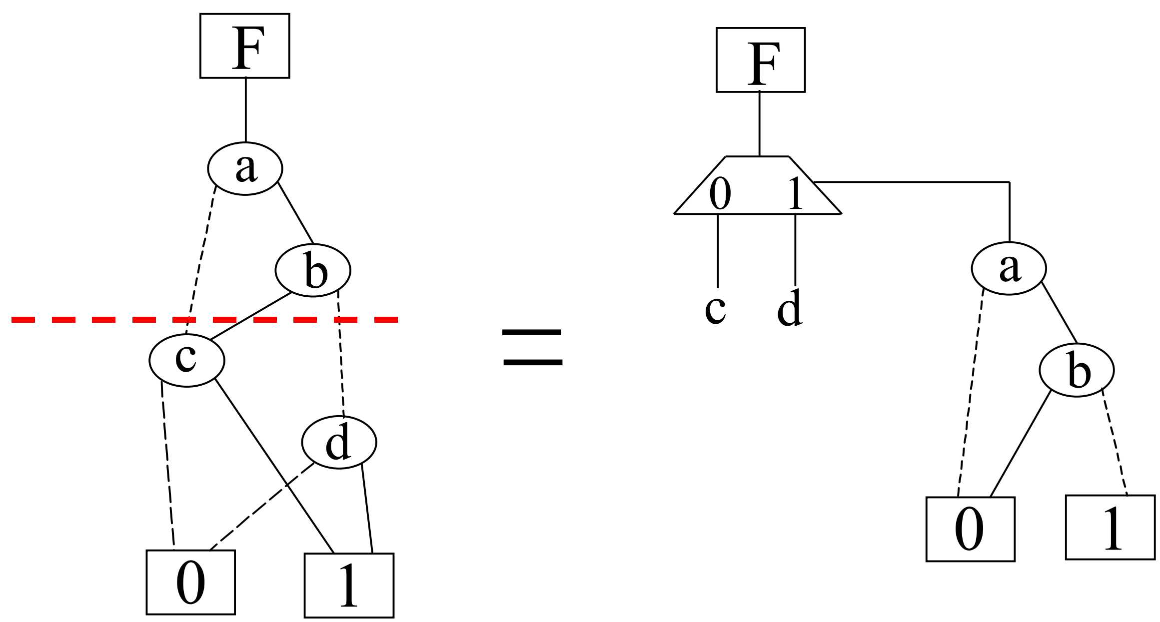


- Complex MUX decomposition

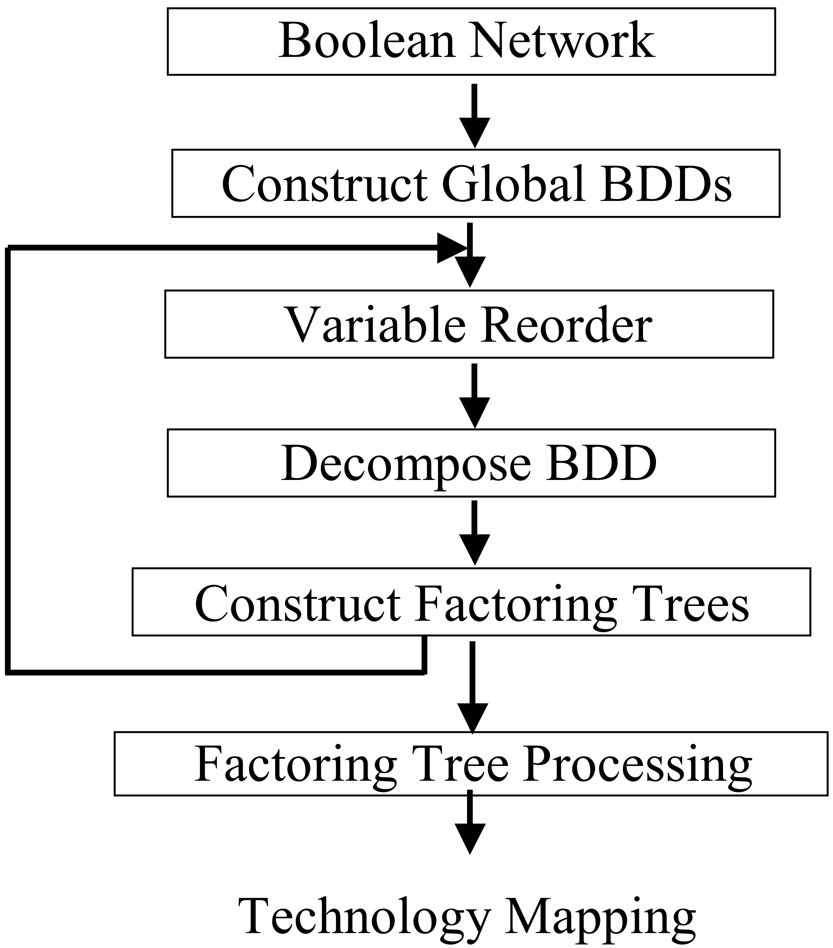


Functional MUX Decomposition

- If two nodes u and v cover all paths in a BDD we can decompose f=hu+h’v where h comes from forcing u=1 and v=0 in F.
  - Useful if u and v share little



Synthesis Flow of BDDlopt

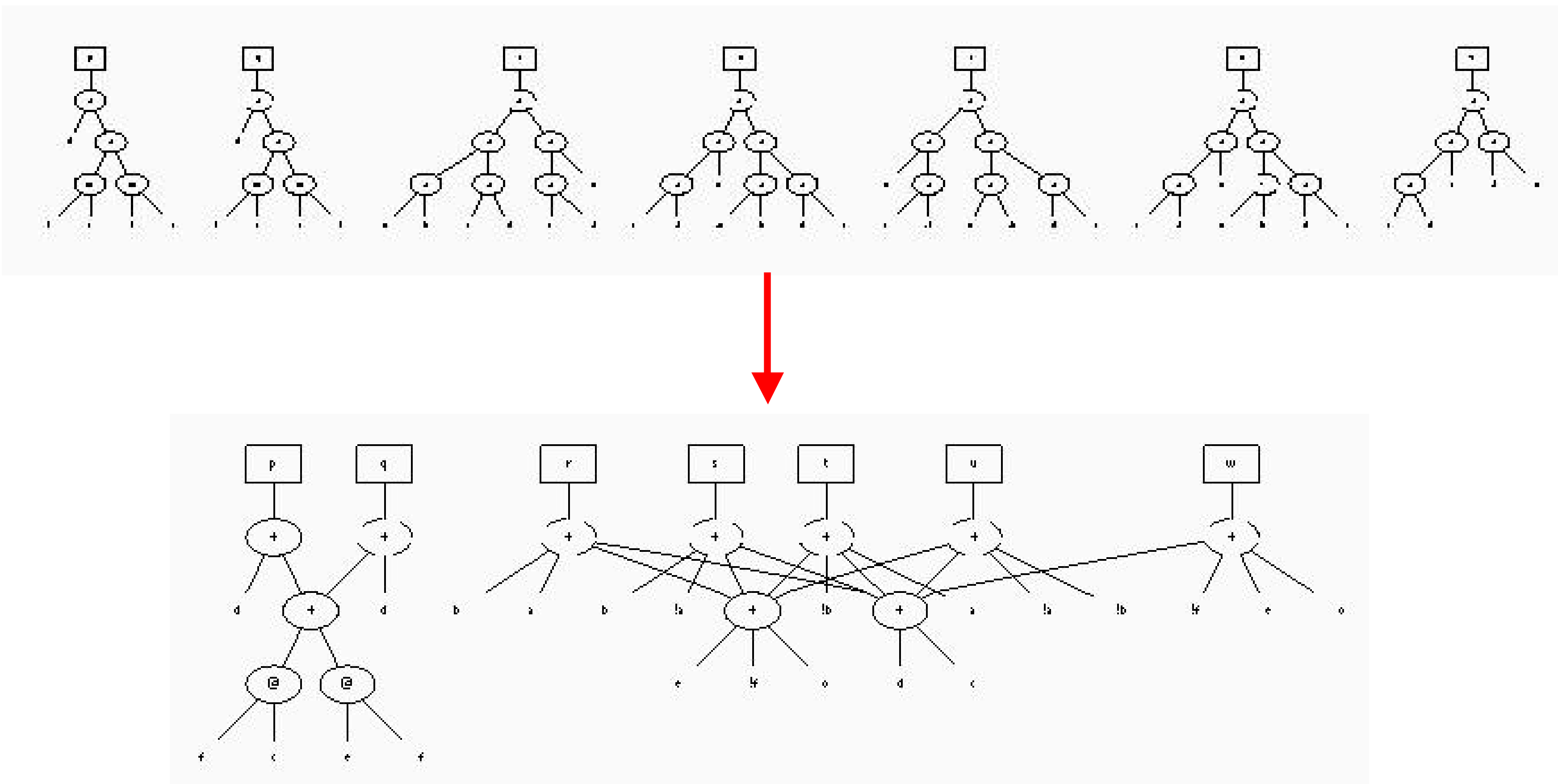


Summary BDD Decomposition

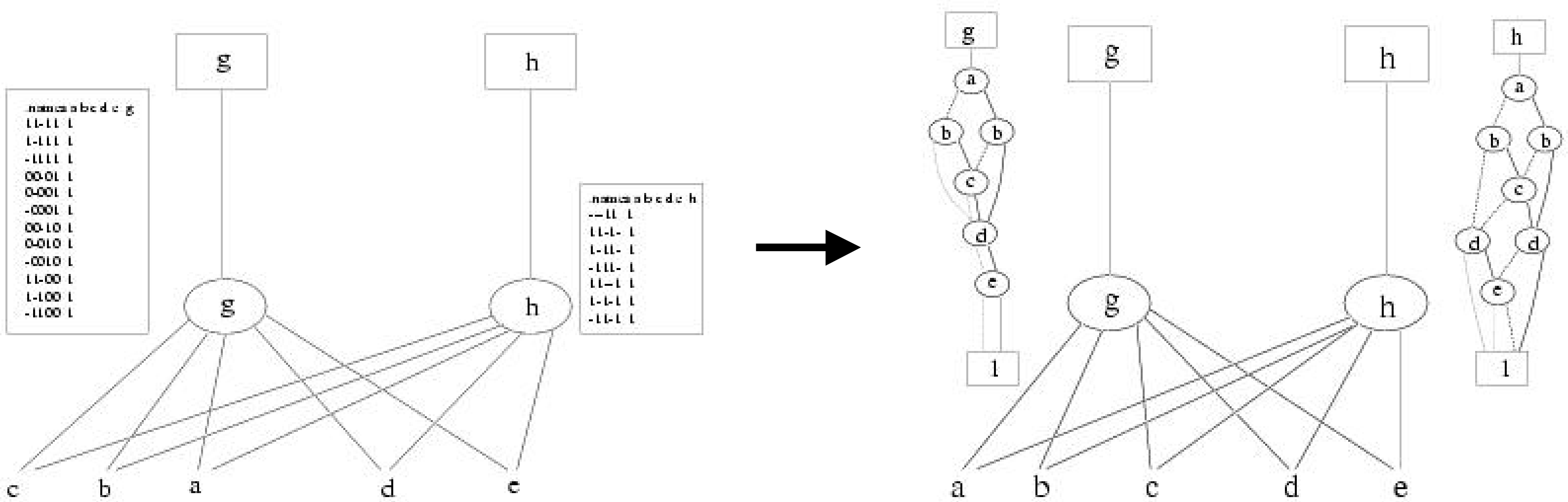
- Ordering allows diagnosis of good cut points
- Cut of BDD Sponsored by redirecting edges of cofactors to constants to minimize the cut portion(s)
- Redirected edges create don’t cares for remaining BDD nodes
- Don’t care minimization by Restrict, Constrain, or GFRC

- Linear Decomposition theorem allows non-binary decompositions as a generalization of cuts.

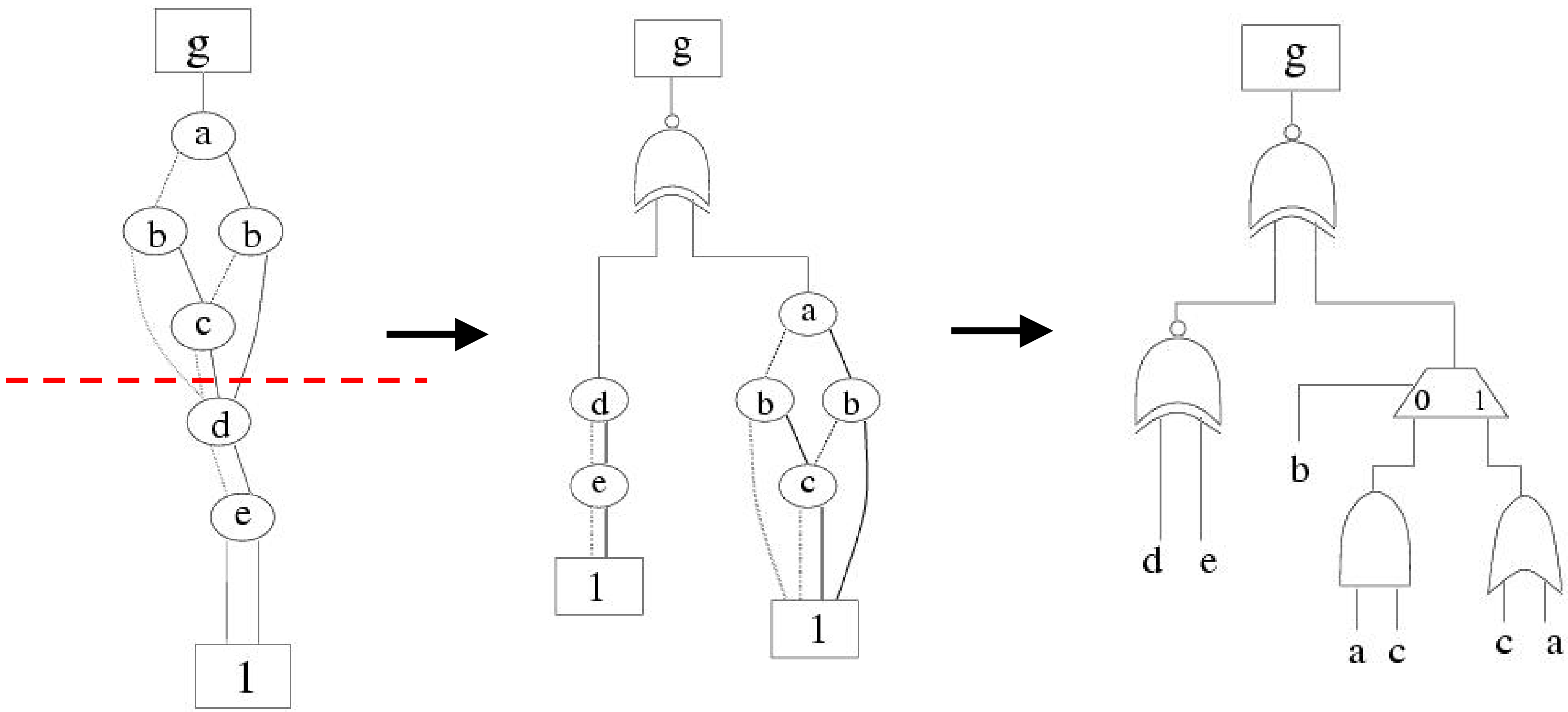
BDD Cut selection



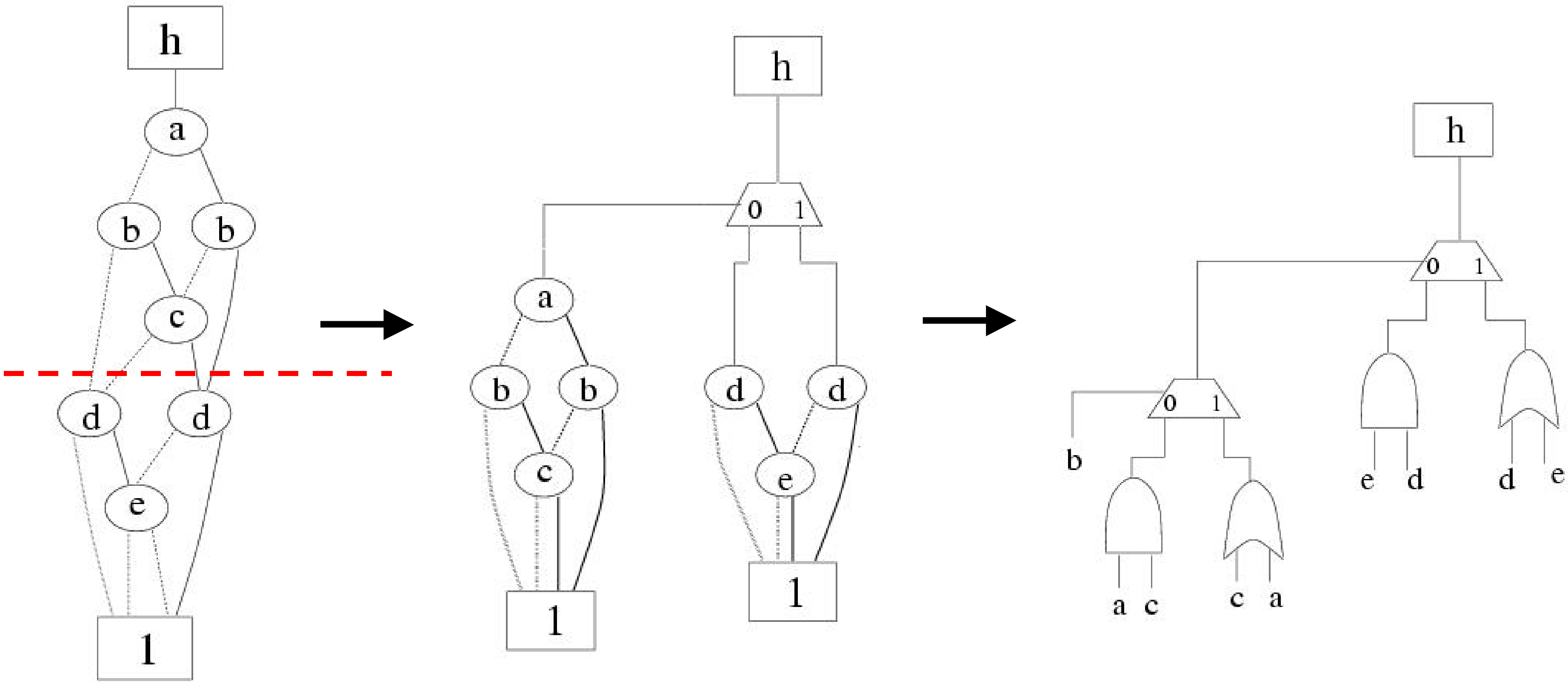
Synthesis Example I



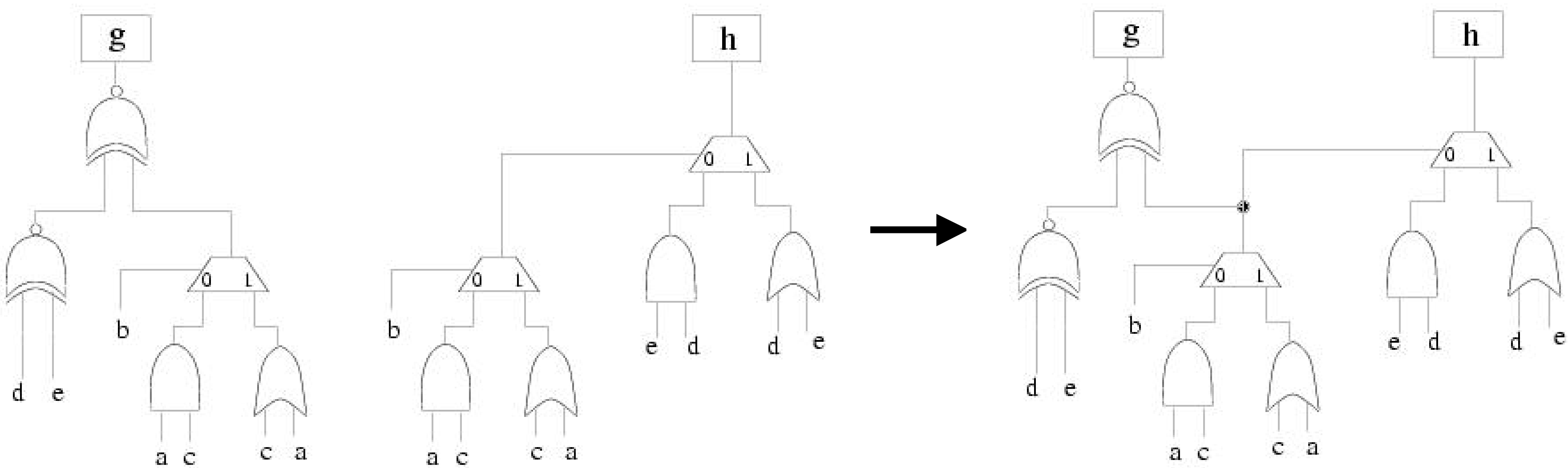
Synthesis Example II (g)



Synthesis Example III (h)

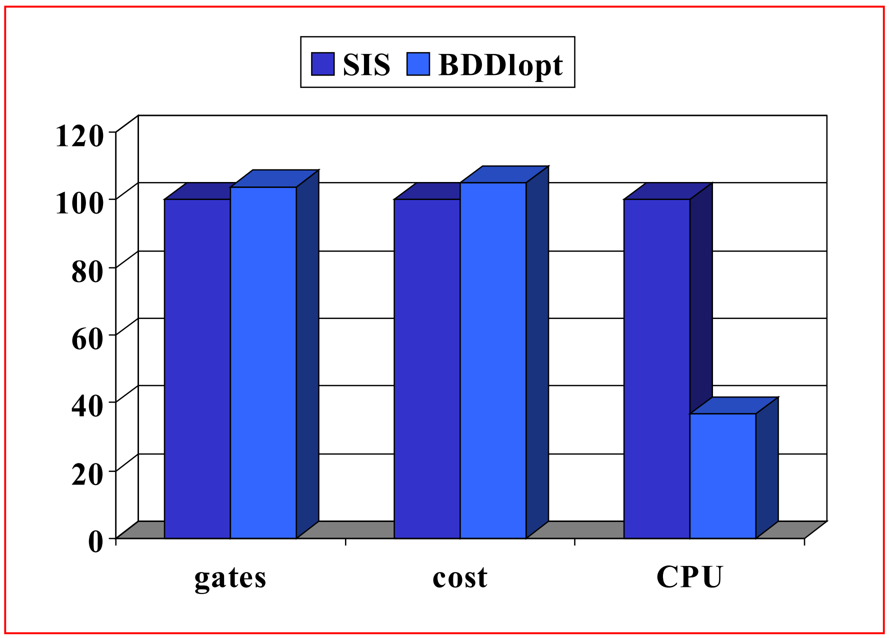


Synthesis Example IV (share)

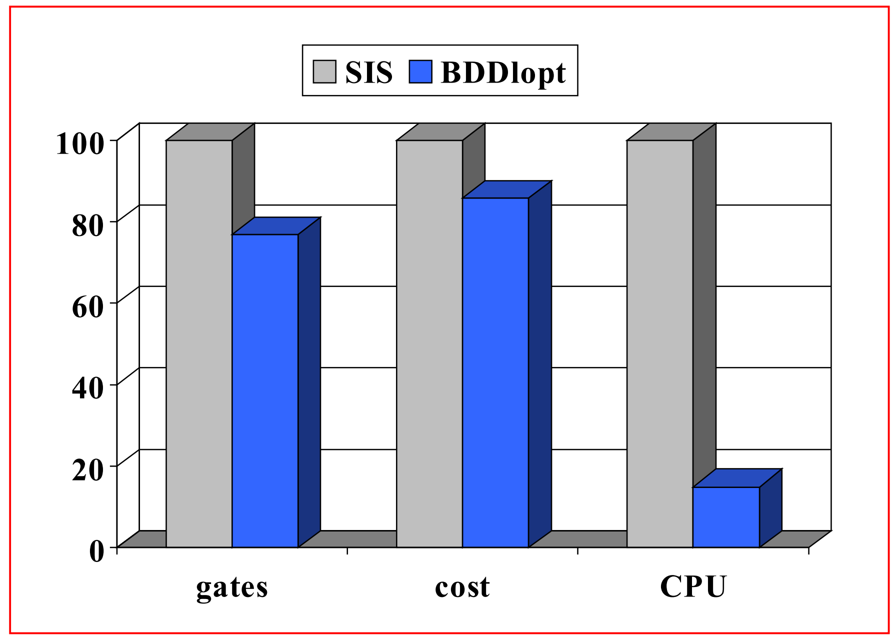


- Keep hash of BDD fragments
  - Equivalence and inverse are trivial since hashes are equal
  - Bias recursive cuts to enhance chance of reuse

Experimental Results



Average ratio of AND/OR-intensive



Average ratio of XOR-intensive

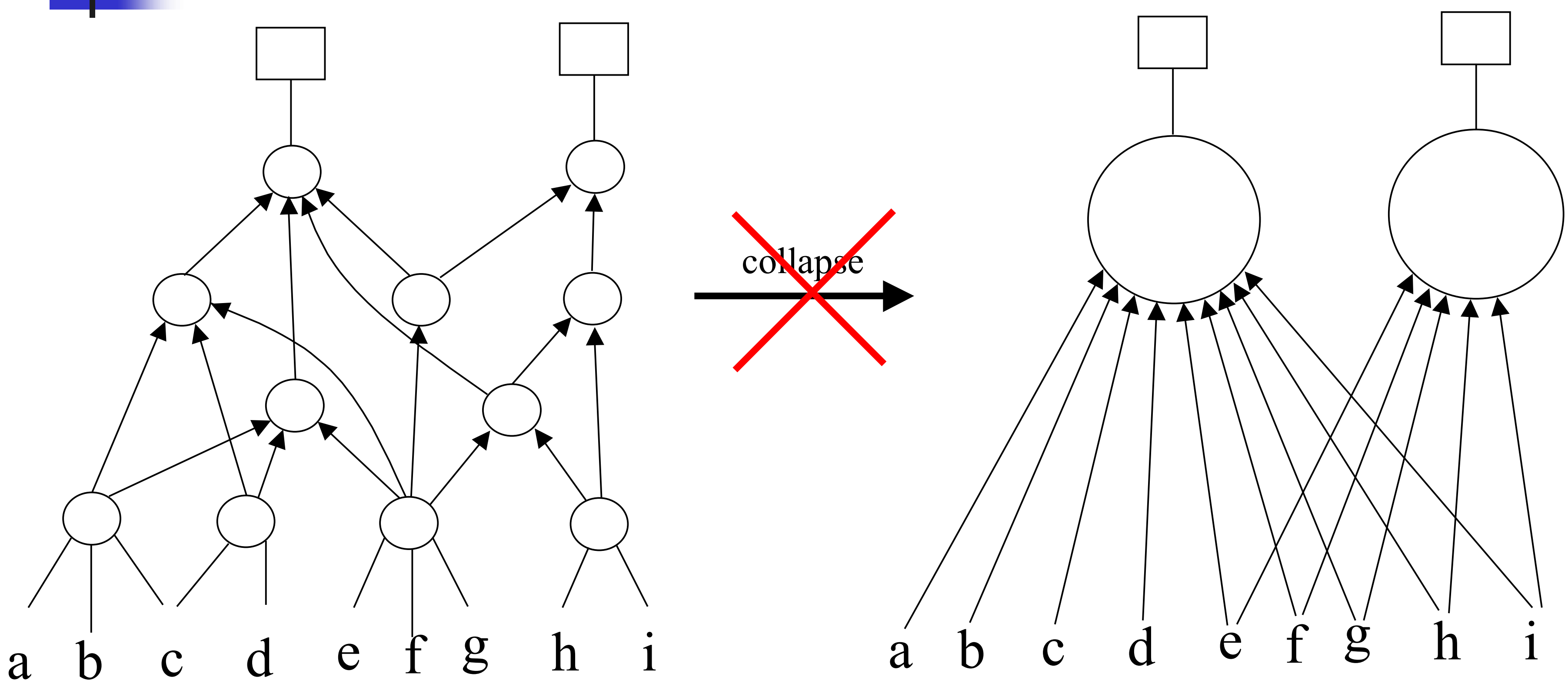
Lessons from BDDlopt

- BDD decomposition is a good alternative to traditional logic optimization.
- BDD decomposition-based logic optimization is fast.
- Stand-alone BDD decomposition is not amenable to large circuits.
  - SCALABILITY

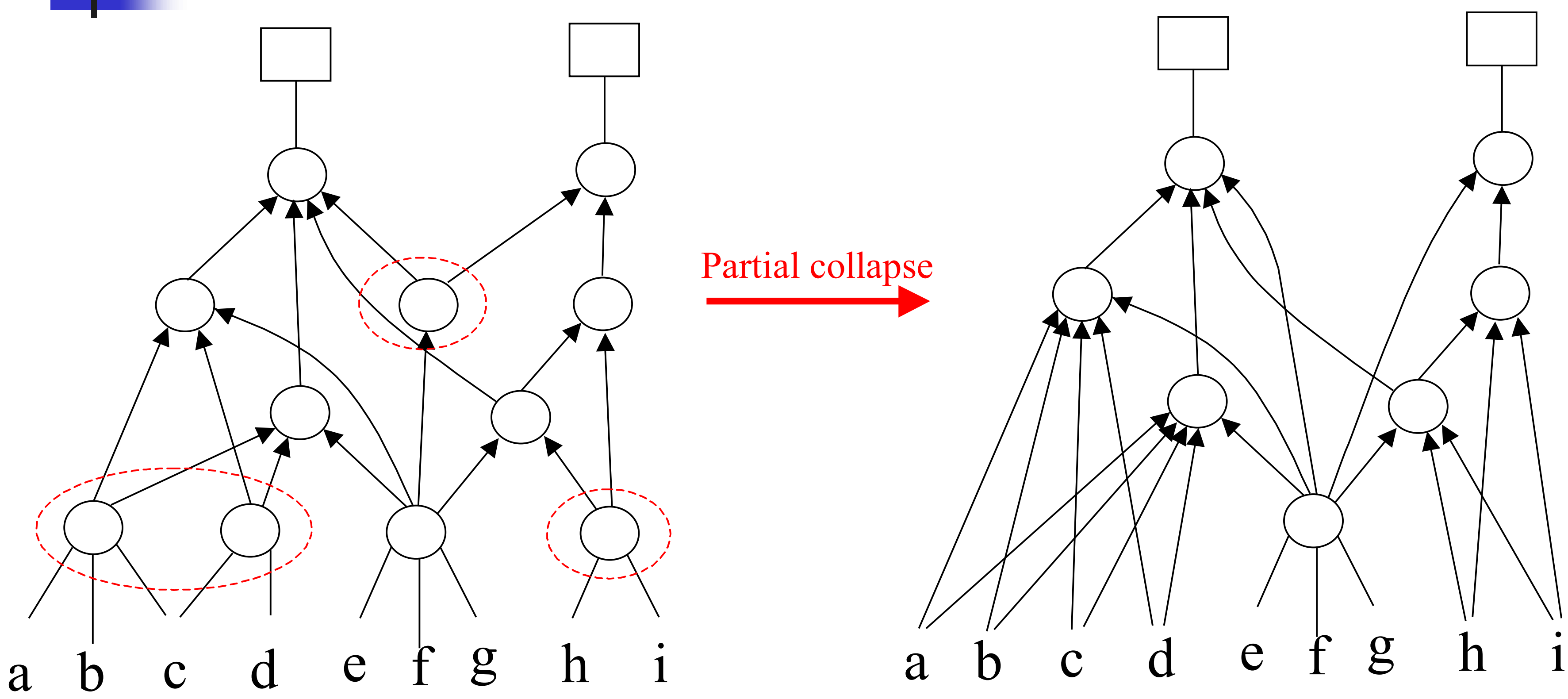
Local vs. Global BDD Representation

Circuits	Global BDDs	Local BDDs
C1355	33450	893
C1908	6734	1229
C2670	5554	1712
C3540	25828	2326
C432	1226	283
C499	26890	341
C5315	2942	3516
C7552	19322	5012
C880	15004	601
pair	4940	1808
rot	7340	934

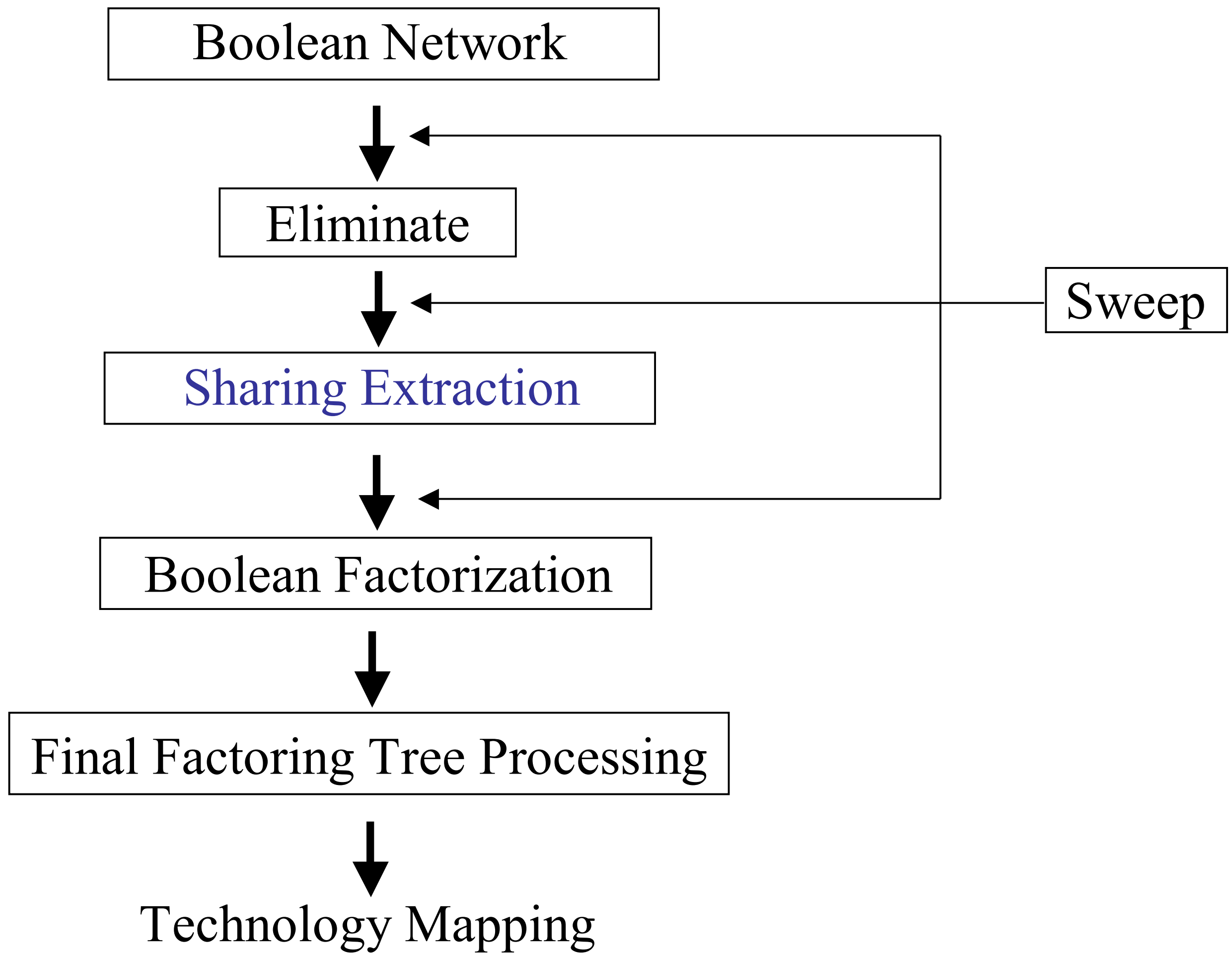
Trade-off Between Local and Global Representation



Trade-off Between Local and Global Representation



BDD-Based Logic Synthesis System (BDS)



Sweep Boolean Network

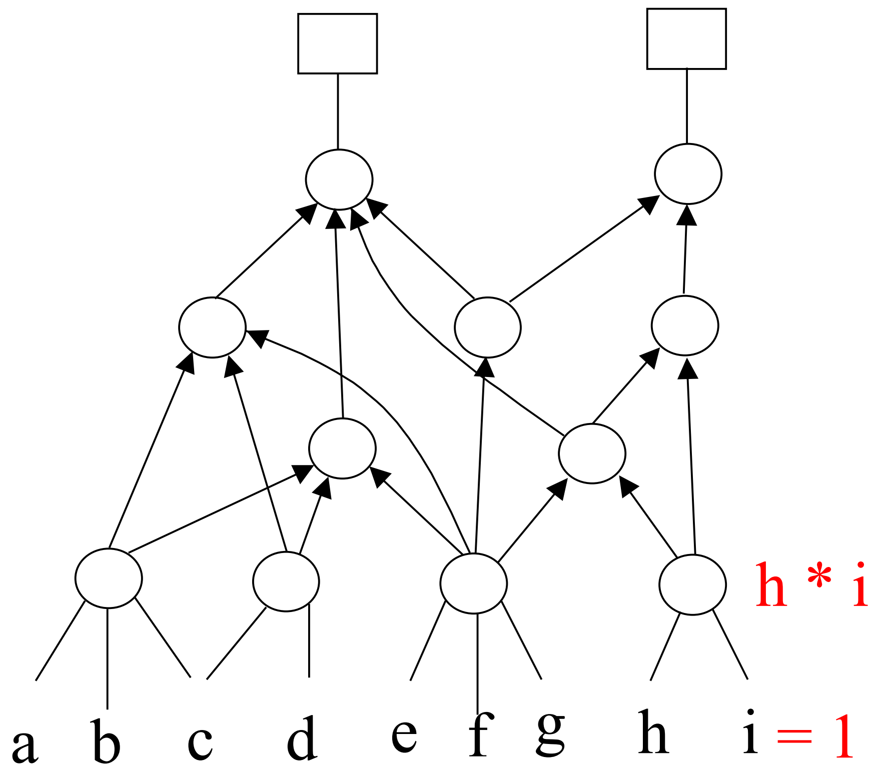
Goal : Network preprocessing.

- Constant propagation. $a + 1 = 1$; $a + a' = 1$; etc

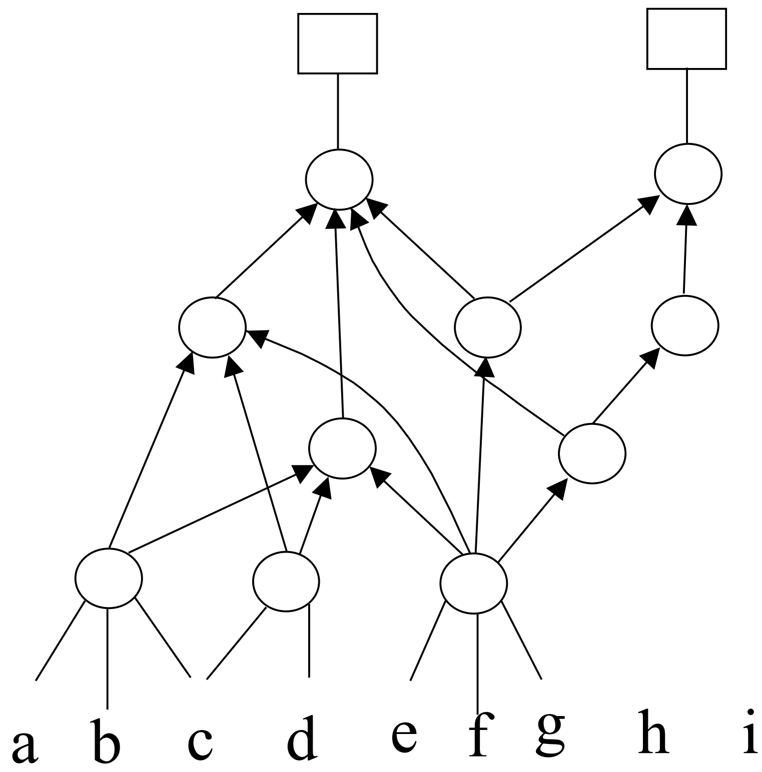
- Remove single-input Boolean nodes.

- Remove functionally duplicated Boolean nodes.

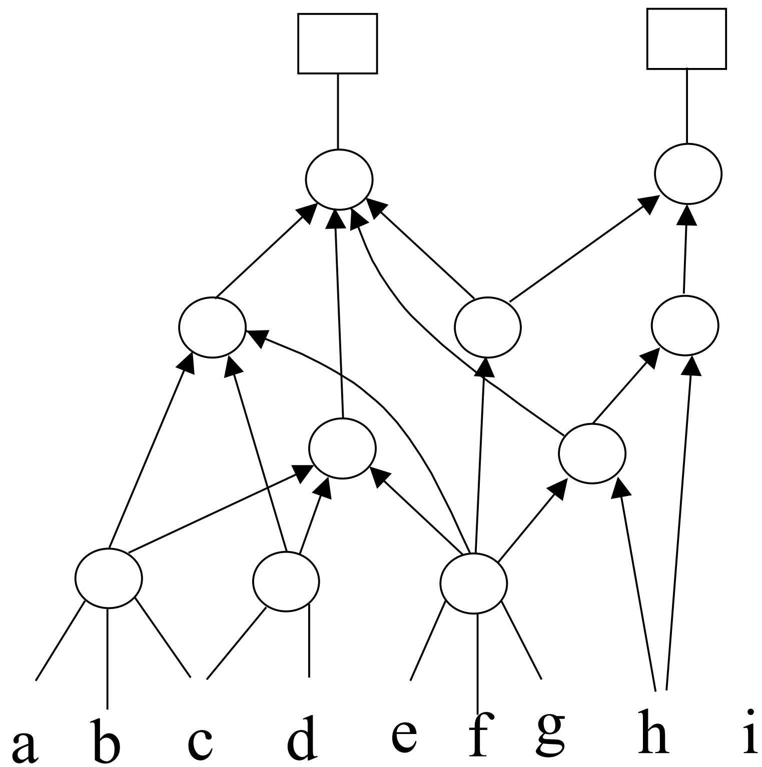
Sweep Boolean Network (e.g.)



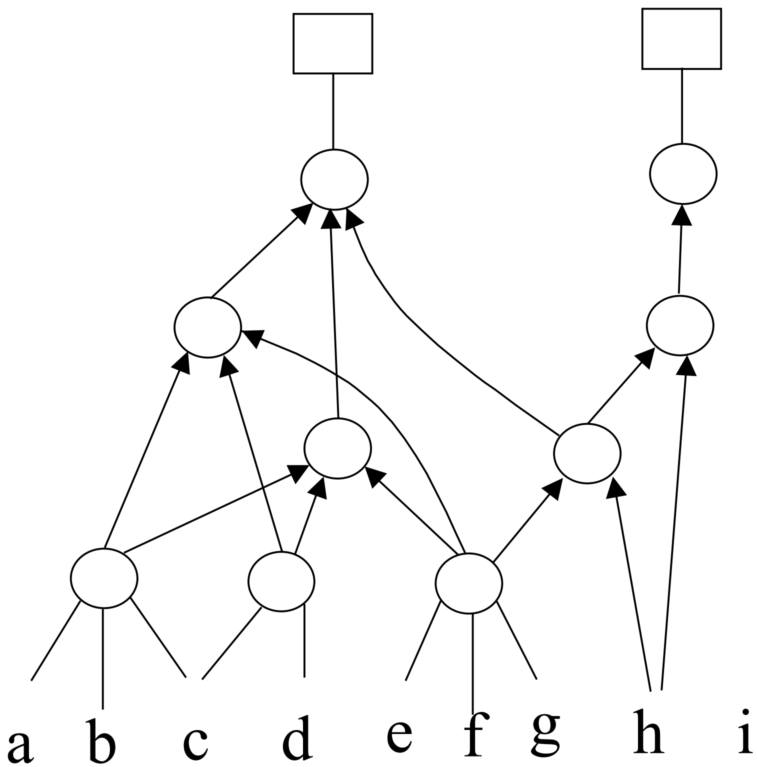
Sweep Boolean Network



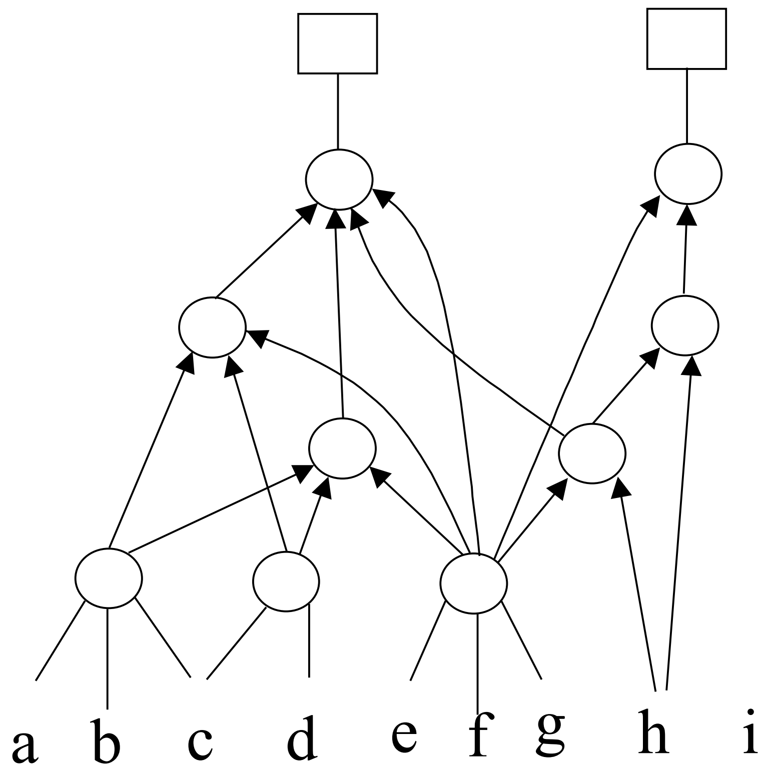
Sweep Boolean Network



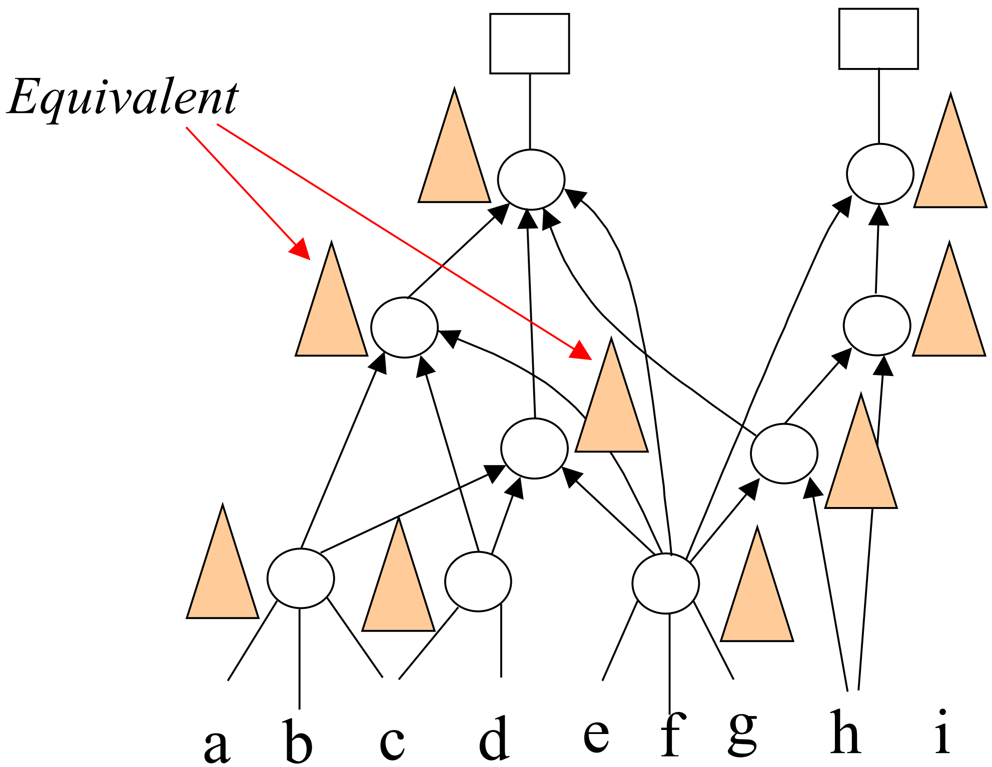
Sweep Boolean Network



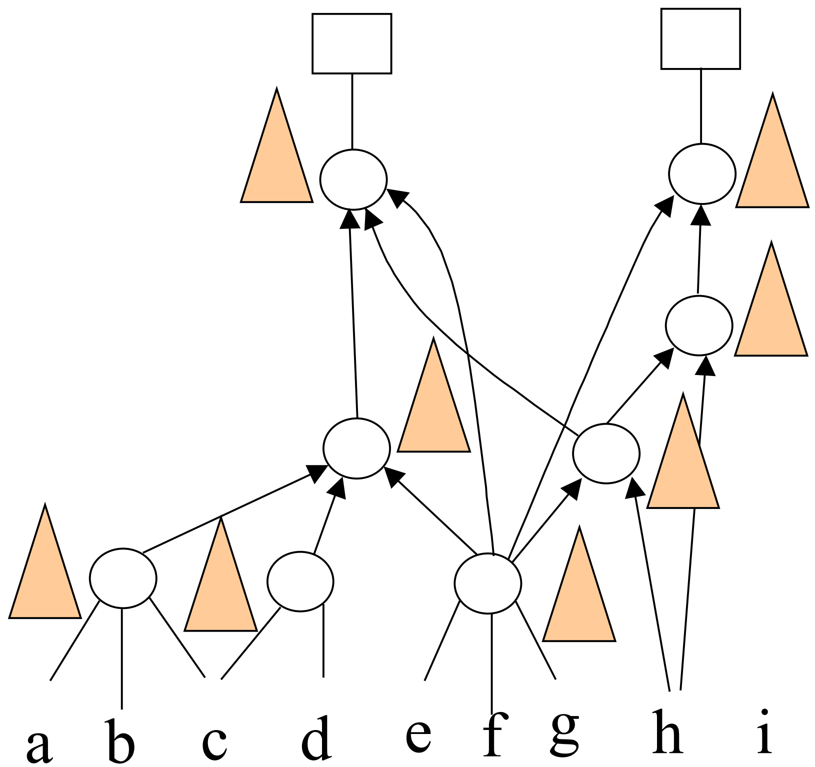
Sweep Boolean Network



Sweep Boolean Network



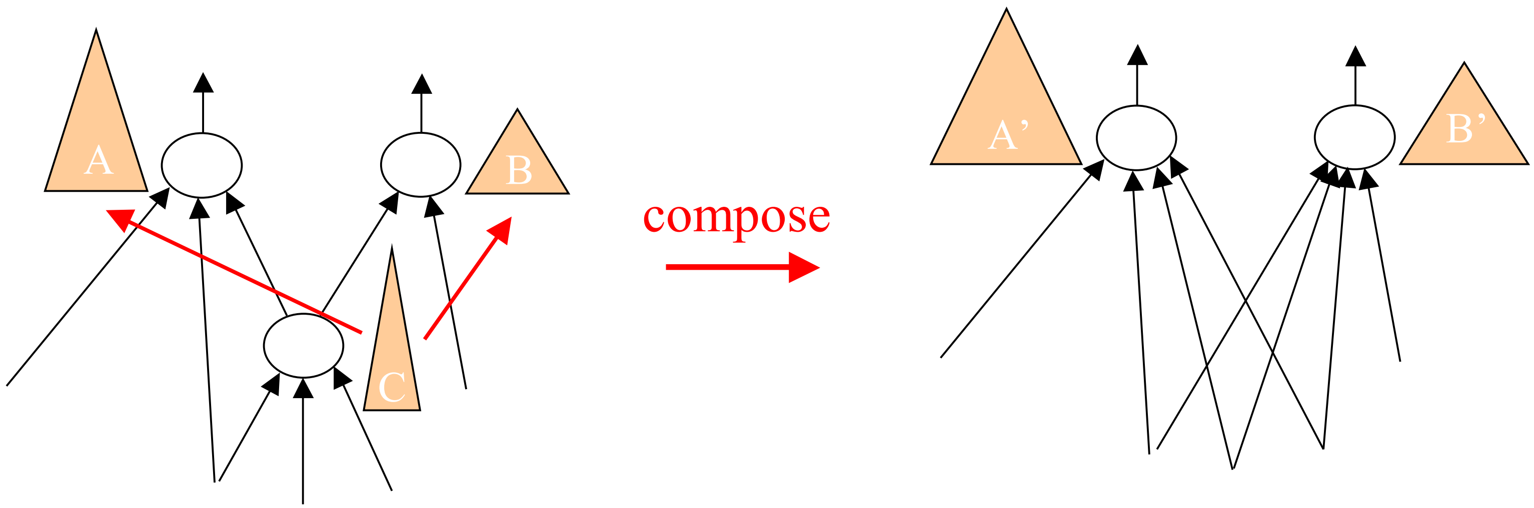
Sweep Boolean Network

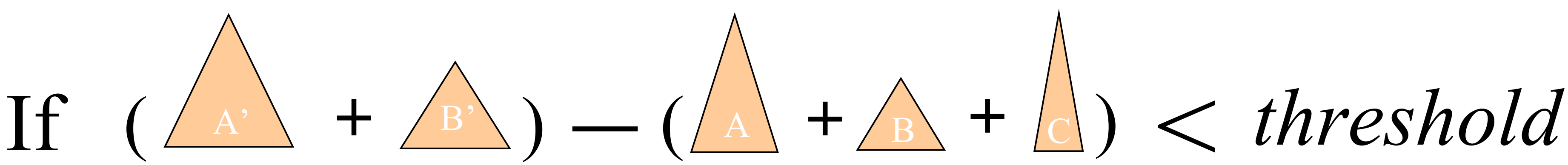


Sweep Duplicated Nodes

Circuits	Total Nodes	Duplicated Nodes
C1908	441	118
C2670	787	72
C3540	956	247
C5315	1467	197
C6228	2353	30
C7552	2165	355
C880	302	10
dalu	985	249
i8	1183	186
i9	329	22
i10	1634	84
pair	830	16
vda	123	3

Boolean Node Elimination





Eliminate Algorithm

Conventional

```
candidate = collect(bddmgr,network);
while(candidate) {
  execute(bddmgr,candidate);
  reorder(bddmgr);
  candidate = collect(bddmgr,network);
}
```

BDS

```
candidate = collect(bddmgr,network);
while(candidate) {
  execute(bddmgr,candidate);
  newbddmgr = bddMapping(bddmgr,network);
  reorder(newbddmgr);
  free(bddmgr);
  bddmgr = newbddmgr;
  candidate = collect(bddmgr,network);
}
```

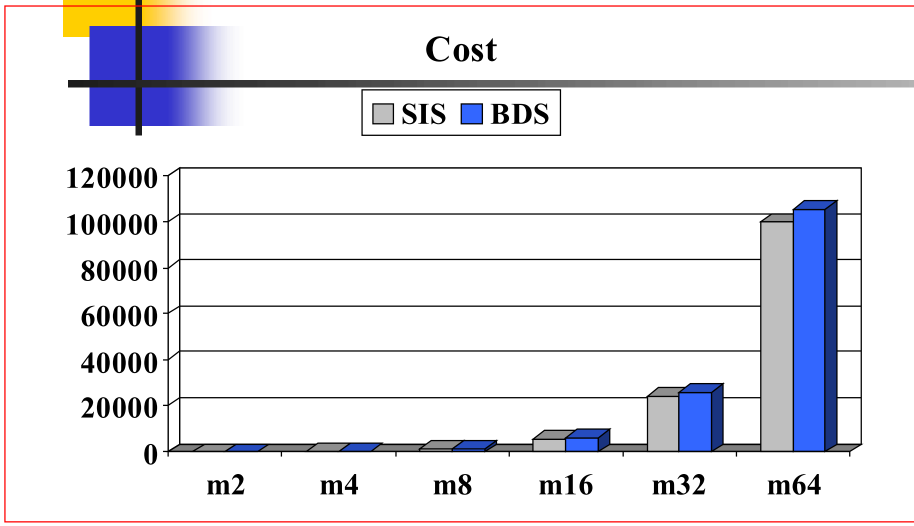
Iterative Eliminate Results

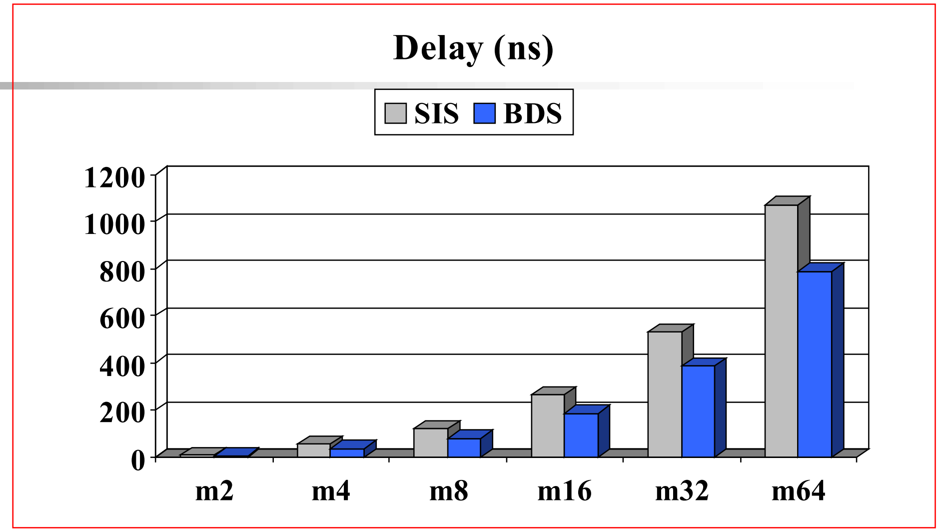
Circuits	Chaudhry *et al* [42]		BDS	
	BDD nodes	CPU (s)	BDD nodes	CPU (s)
C1355	211	270	207	0.3
C1908	310	25.4	276	0.6
C2670	615	197	527	1.7
C3540	974	101.7	901	3.2
C432	181	4.5	183	0.5
C499	196	2.4	228	0.2
C5315	1008	307.6	918	4.0
C6288	1677	540.7	1507	4.4
C7552	1592	382.1	1227	6.4
C880	298	7.5	300	0.4
Total	7066	1838.9	6274	21.7

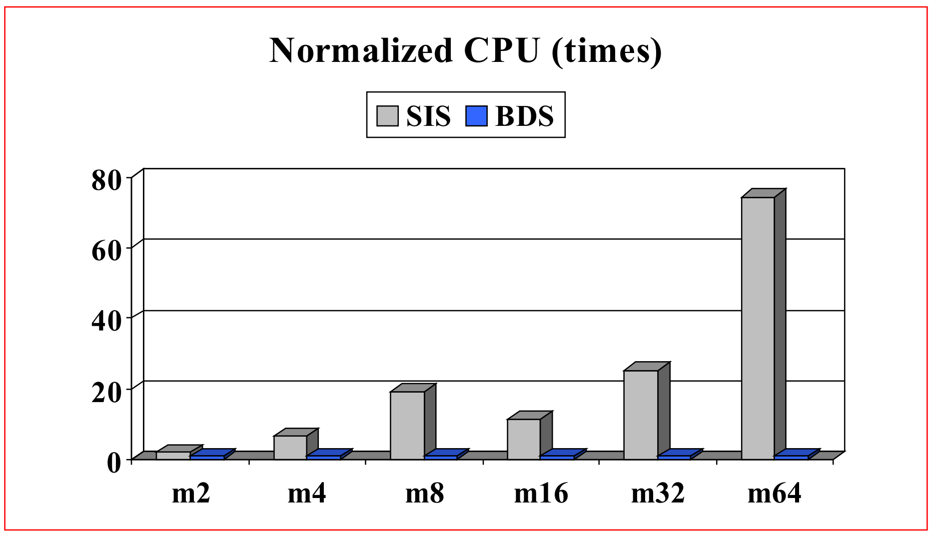
BDS Experimental Results

Circuits	SIS				BDS			
	Area	Delay	CPU (s)	Mem (M)	Area	Delay	CPU (s)	Mem (M)
C1355	689	39.40	6.6	1.2	711	45.60	0.4	1.0
C1908	695	68.60	8.1	1.2	730	65.00	0.8	1.0
C3540	1695	81.40	16.1	3.3	1713	81.20	3.6	1.9
C432	290	75.90	46.1	0.7	357	78.40	0.2	0.5
C499	689	39.40	6.8	0.9	708	43.60	0.6	0.5
C5315	2286	68.60	10.2	3.1	2402	70.50	5.3	3.0
C6288	4631	237.8	21.8	4.1	4677	178.3	3.8	1.1
C7552	3038	115.70	54.2	4.9	3112	83.30	4.2	4.8
C880	567	56.10	1.9	1.0	563	43.20	0.7	0.8
pair	2274	74.30	16.1	2.5	2466	52.60	2.1	2.0
rot	965	51.60	4.5	2.0	1025	51.90	1.0	0.9
Total	17819	908.8	192.4	24.9	18464	793.6	22.7	17.5

Synthesis of Multipliers

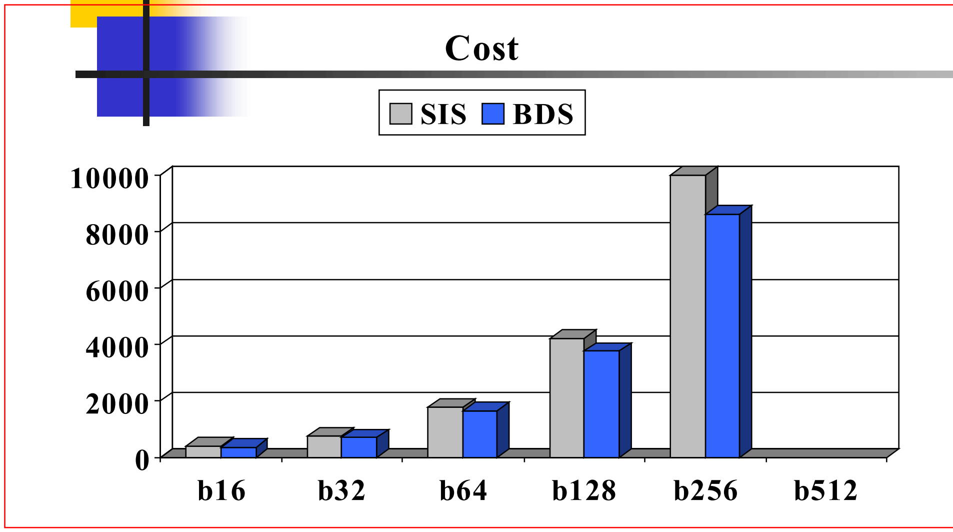


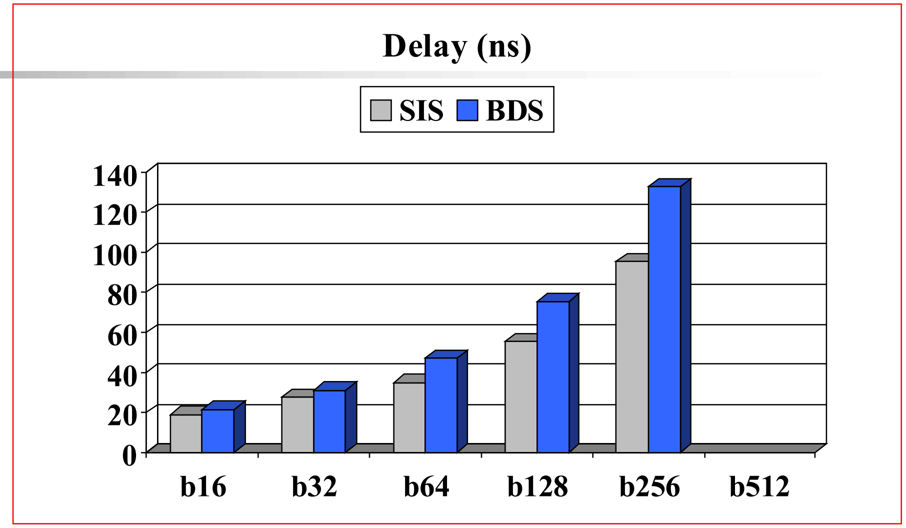


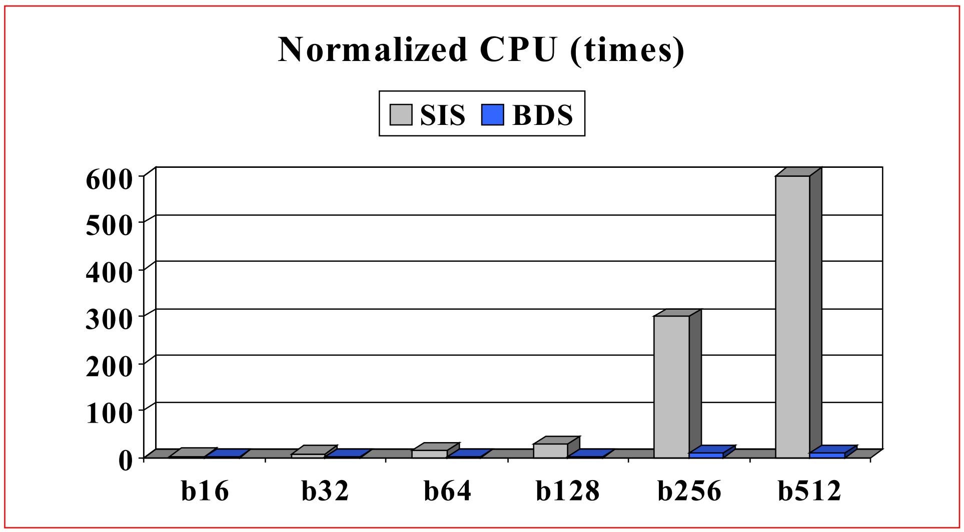


Result of *m64X64*
Size : ~ 41K
CPU (SIS) : 6.6 hrs
CPU (BDS): 5.3 minutes

Synthesis Results for bshift







Result of *bshift512*
Size : ~ 7K
CPU (SIS) : > 15 hrs
CPU (BDS): 1.5 minutes

Synthesis for Mixed CMOS/PTL Logic

CMOS

- Concise AND, OR logic
- High noise immunity
- Level-restoring capability

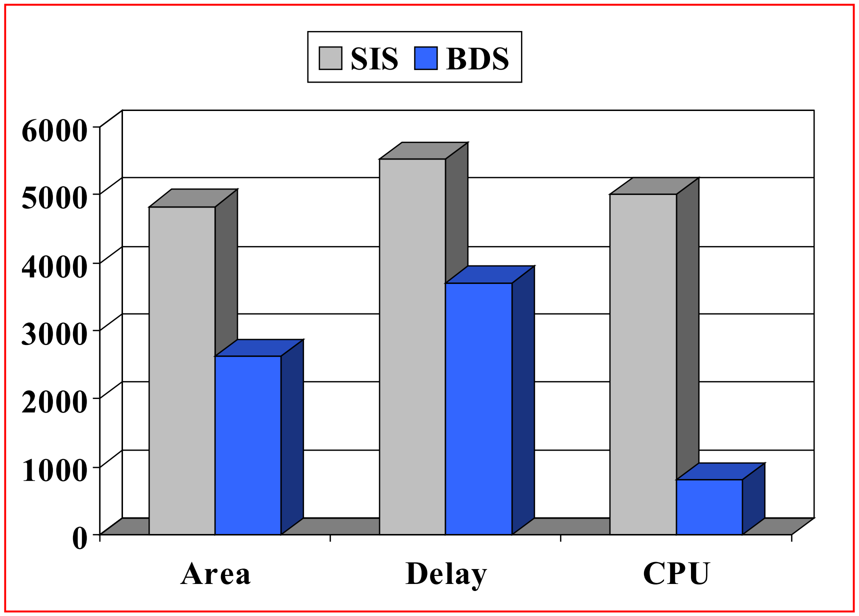
- Inefficient MUX, XOR
- Higher power

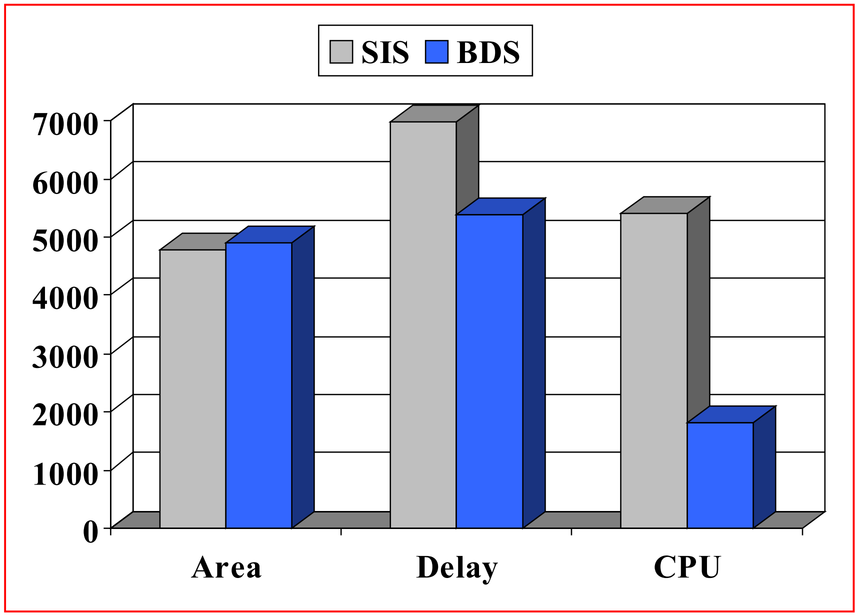
PTL

- Concise MUX, XOR logic
- Low power
- Small area (faster)

- Inefficient AND, OR
- Low noise immunity

Results of Mixed CMOS/PTL Synthesis





XOR-intensive functions

AND/OR-intensive functions

Conclusions

- New theory of BDD decomposition. For the first time BDD is used to perform Boolean AND/OR, XOR decompositions.
- First unified approach to both AND/OR- and XOR-intensive Boolean functions.
- Implementation of first working BDD-based multi-level logic synthesis system (BDS).
- Efficient BDD manipulation techniques.
- First systematic approach to mixed CMOS/PTL synthesis