PORTLAND QUANTUM LOGIC GROUP

PART TWO

Marek Perkowski

Absolutely Minimum Background on Binary Decision Diagrams (BDD) and Kronecker Functional Decision Diagrams

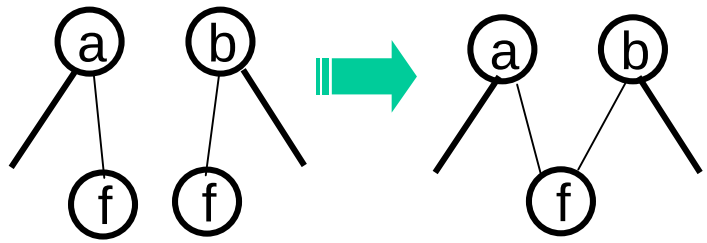
- BDDs are based on recursive Shannon expansion

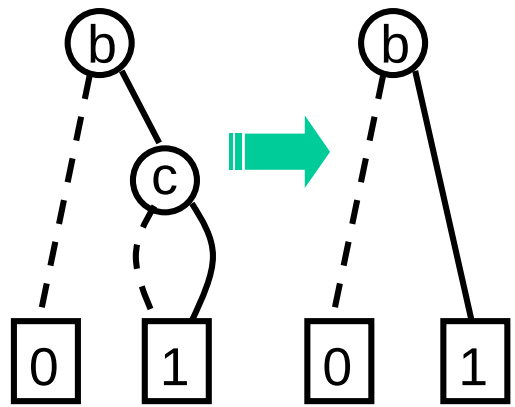
$$F = x F_x + x' F_{x'}$$

- Compact data structure for Boolean logic
  - can represents sets of objects (states) encoded as Boolean functions
- Canonical representation
  - reduced ordered BDDs (ROBDD) are canonical
  - essential for simulation, analysis, synthesis and verification

BDD Construction

- Typically done using *APPLY* operator
- Reduction rules
  - remove duplicate terminals
  - merge duplicate nodes (*isomorphic* subgraphs)
  - remove *redundant* nodes
- Redundant nodes:
  - nodes with identical children





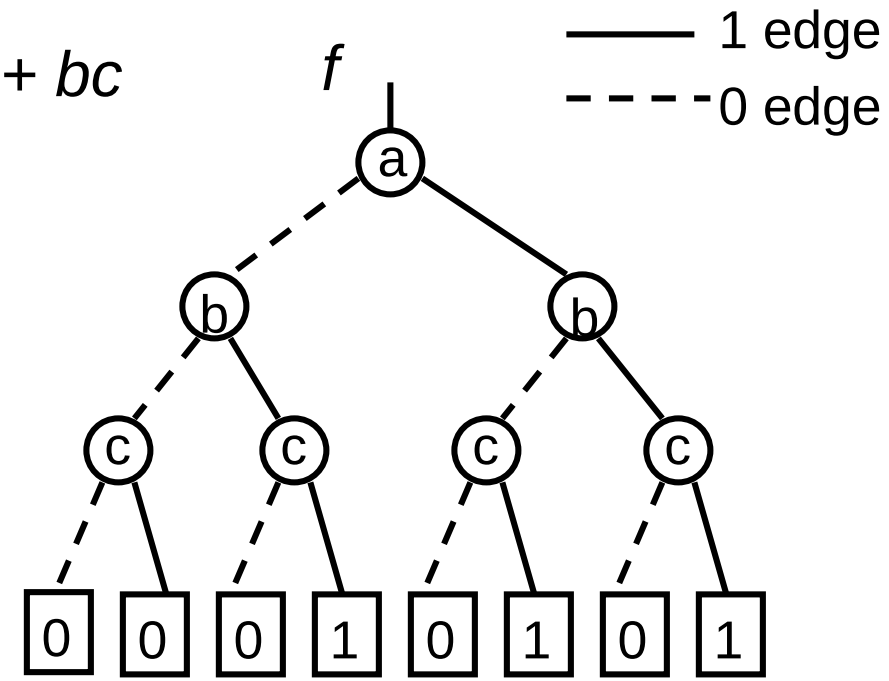
BDD Construction – your first BDD

- Construction of a Reduced Ordered BDD

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Truth table

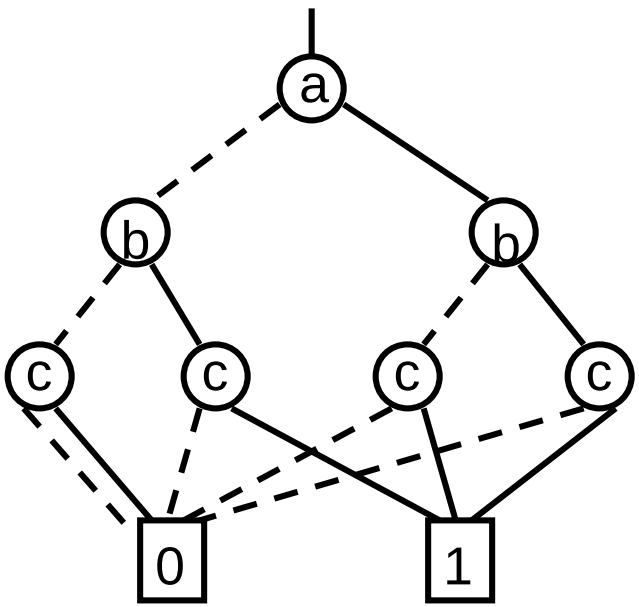
$f = ac + bc$



Decision tree

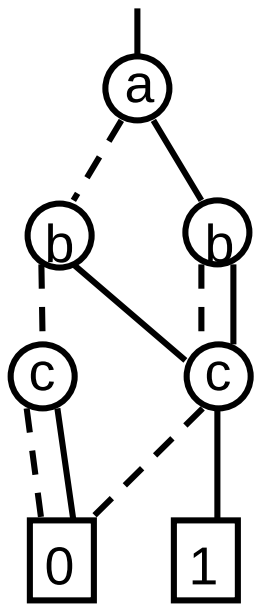
BDD Construction – cont'd

f



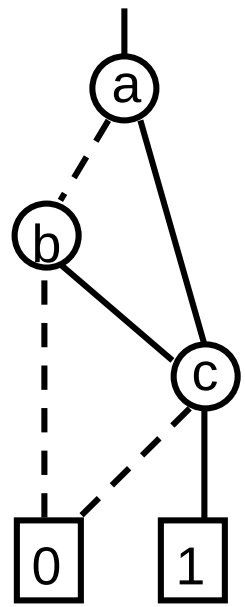
1. Remove duplicate terminals

f



2. Merge duplicate nodes

f = (a+b)c



3. Remove redundant nodes

Decomposition types

$$f = \overline{x}_i f_i^0 + x_i f_i^1 \qquad \text{Shannon (S)} \qquad (1)$$

$$f = f_i^0 \oplus x_i f_i^2 \qquad \text{positive Davio (pD)} \qquad (2)$$

$$f = f_i^1 \oplus \overline{x}_i f_i^2 \qquad \text{negative Davio (nD)} \qquad (3)$$

where $f_i^0 (f_i^1)$ denotes the *cofactor* of f with respect to $x_i = 0$ $(x_i = 1)$, and f_i^2 is defined as $f_i^2 := f_i^0 \oplus f_i^1$, $\oplus$ being the exclusive OR operation.

Decomposition types are associated to the variables in *Xn* with the help of a decomposition type list (**DTL**) d:=(d1,…,dn) where di ∈{ S, pD, nD}

KFDD

- Definition

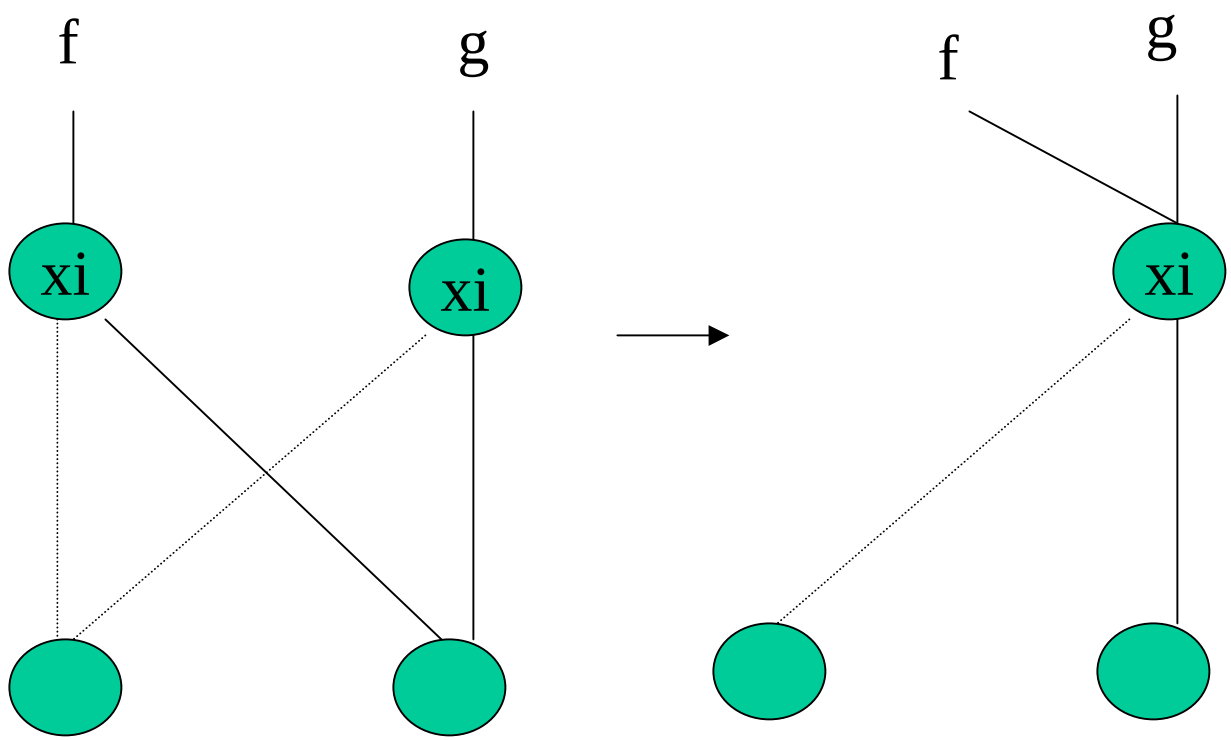
1) If G consists of a single node labeled with 0 (1), then G is a KFDD for the constant 0 (1) function.
2) If G has a root v with label x_i, then G is a KFDD for

$$\begin{cases} \overline{x}_i f_{\mathrm{low(v)}} + x_i f_{\mathrm{high(v)}} : & d_i = \mathrm{S} \\ f_{\mathrm{low(v)}} \oplus x_i f_{\mathrm{high(v)}} : & d_i = \mathrm{pD} \\ f_{\mathrm{low(v)}} \oplus \overline{x}_i f_{\mathrm{high(v)}} : & d_i = \mathrm{nD} \end{cases}$$

where $f_{\mathrm{low(v)}}$ ($f_{\mathrm{high(v)}}$) is the function represented by the KFDD rooted at $low(v)(high(v))$.

Three different reductions types

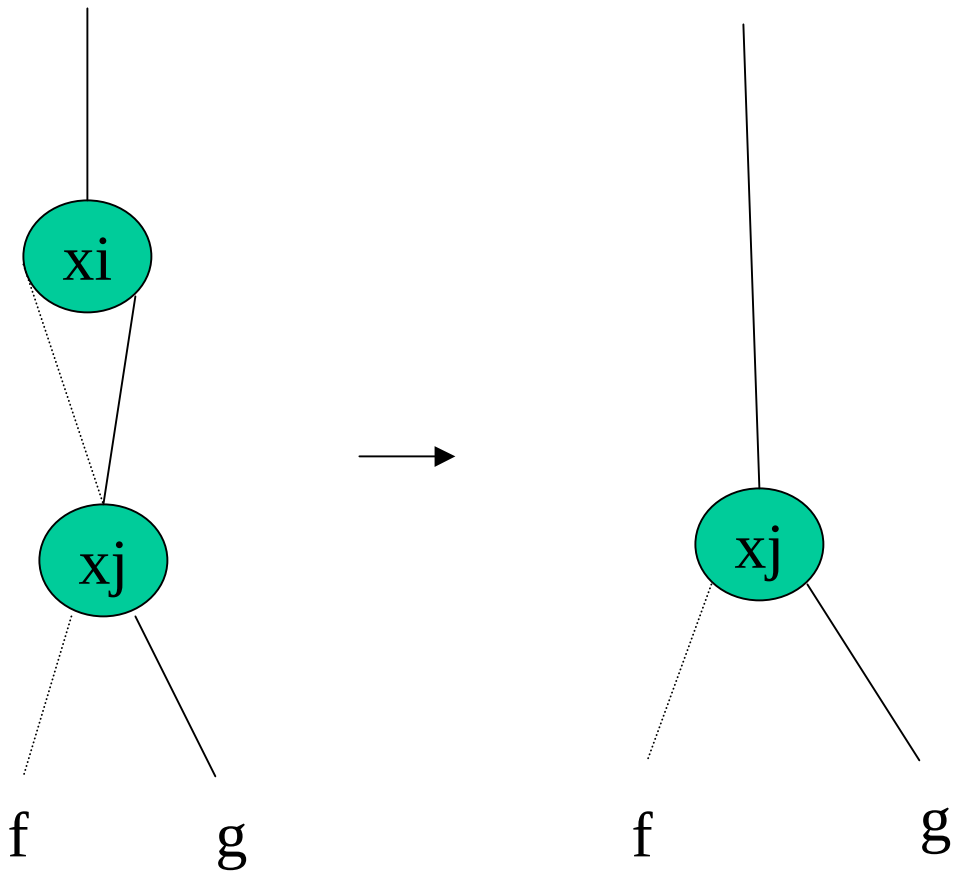
Type I :



Each node in a DD is a candidate for the application

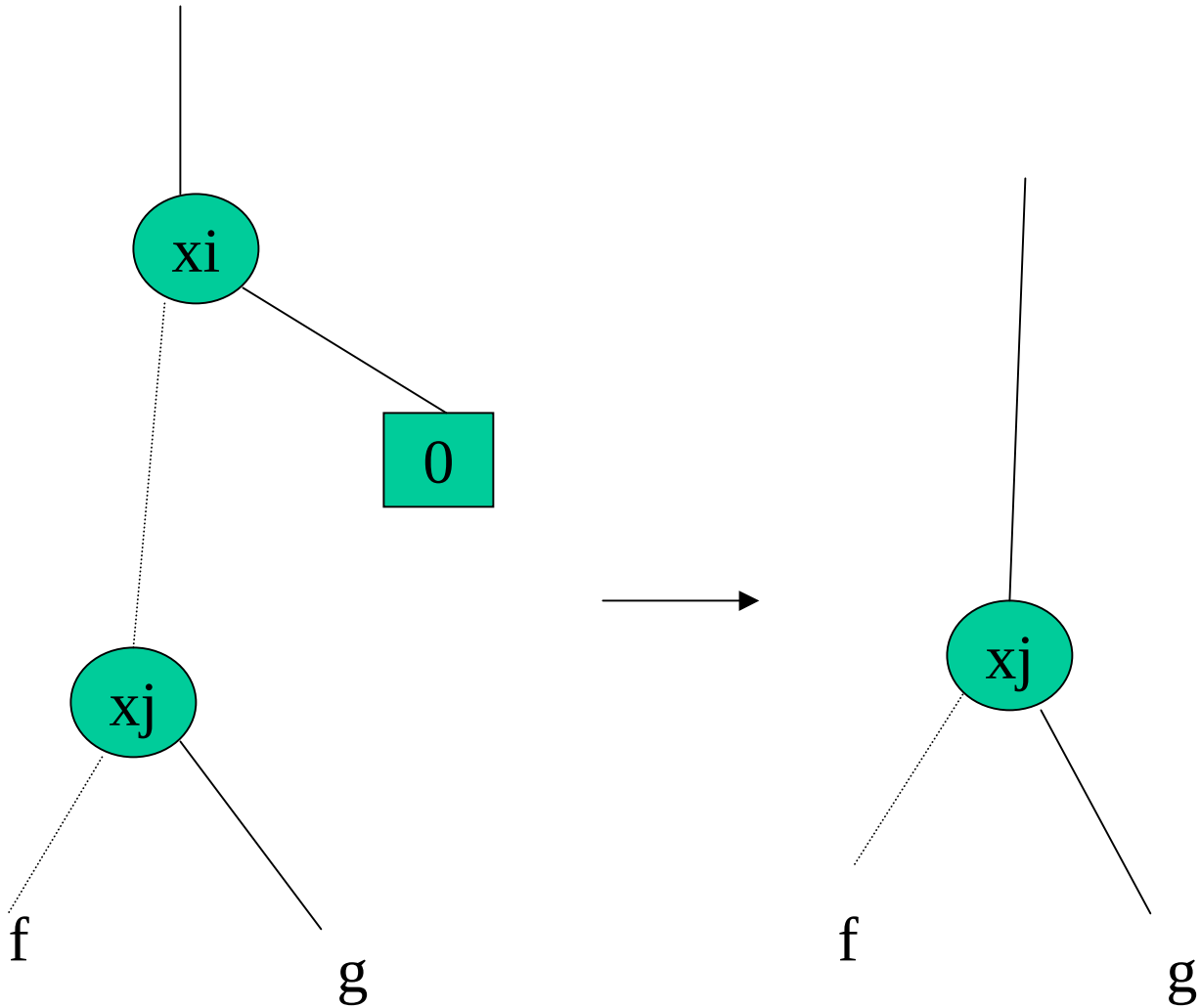
Three different reductions types (cont'd)

Type S

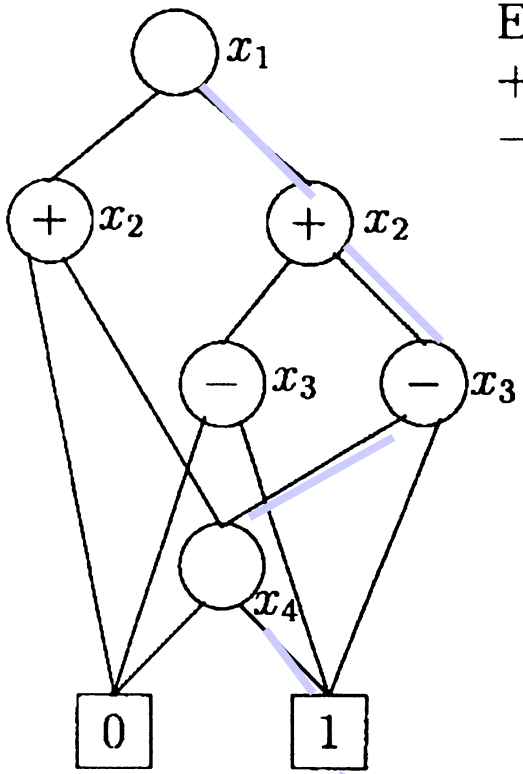


Three different reductions types (cont'd)

Type D



Example for OKFDD



Empty nodes denote S-nodes
+ denotes pD-nodes
− denotes nD-nodes

Fig. 2. Example for OKFDD.

$$F = x_1x_2x_4 \oplus x_1x_2\bar{x}_3 \oplus x_1\bar{x}_3 \oplus \bar{x}_1x_2x_4$$

Example for OKFDD (cont'd)

$$\bar{x}_1(x_2x_4)+x_1(x_2x_4\oplus x_2\bar{x}_3\oplus x_3)$$

X1 S-node

$$(0)\oplus x_2(x_4)\qquad(\bar{x}_3)\oplus x_2(\bar{x}_3\oplus x_4)$$

X2 pD-node

$$(0)\oplus \bar{x}_3(1)\qquad(x_4)\oplus \bar{x}_3(1)$$

X3 nD-node

Example for OKFDD's with different DTL's

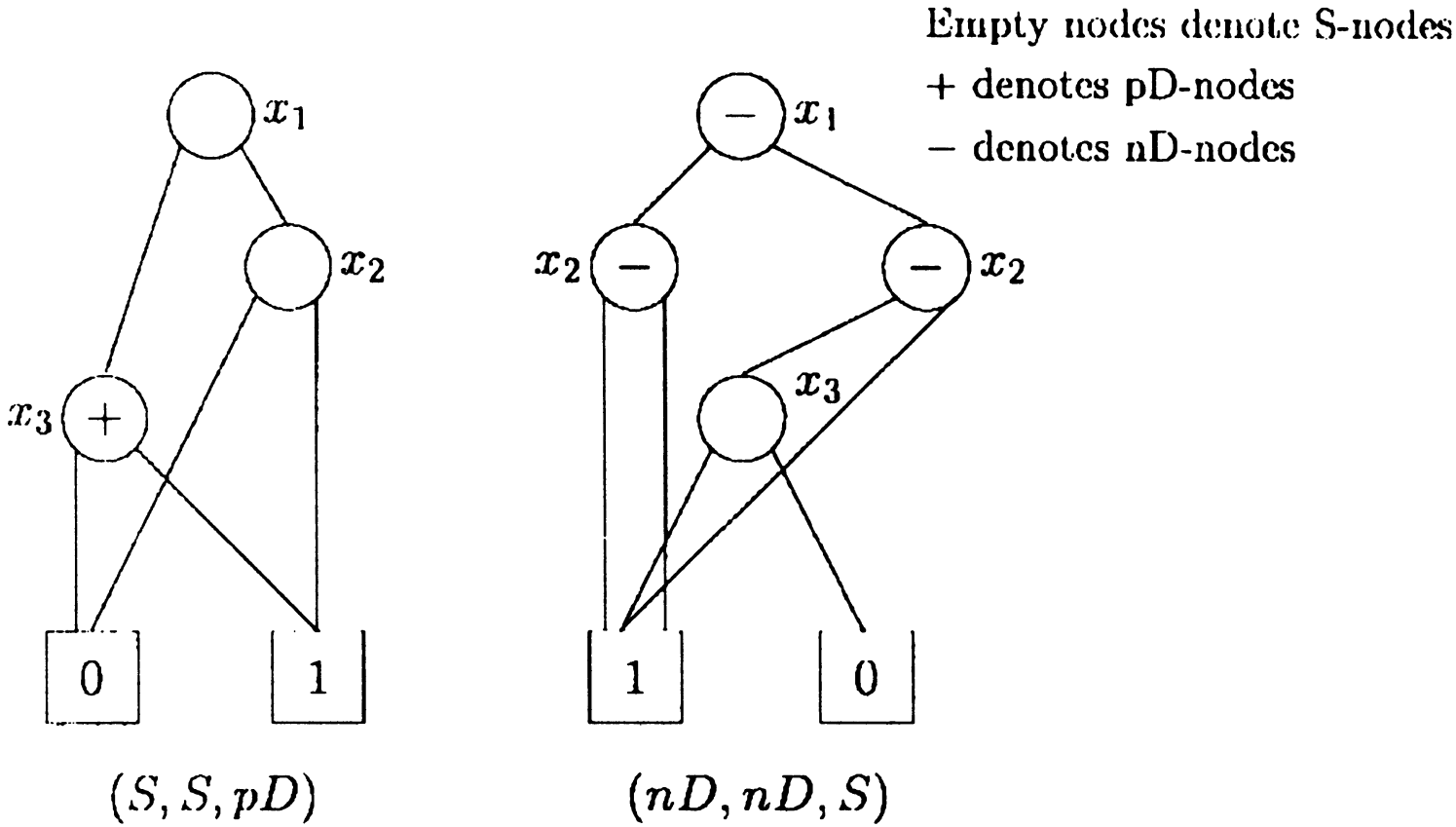


Fig. 3. Example for OKFDD's with different DTL's.

Complement Edges

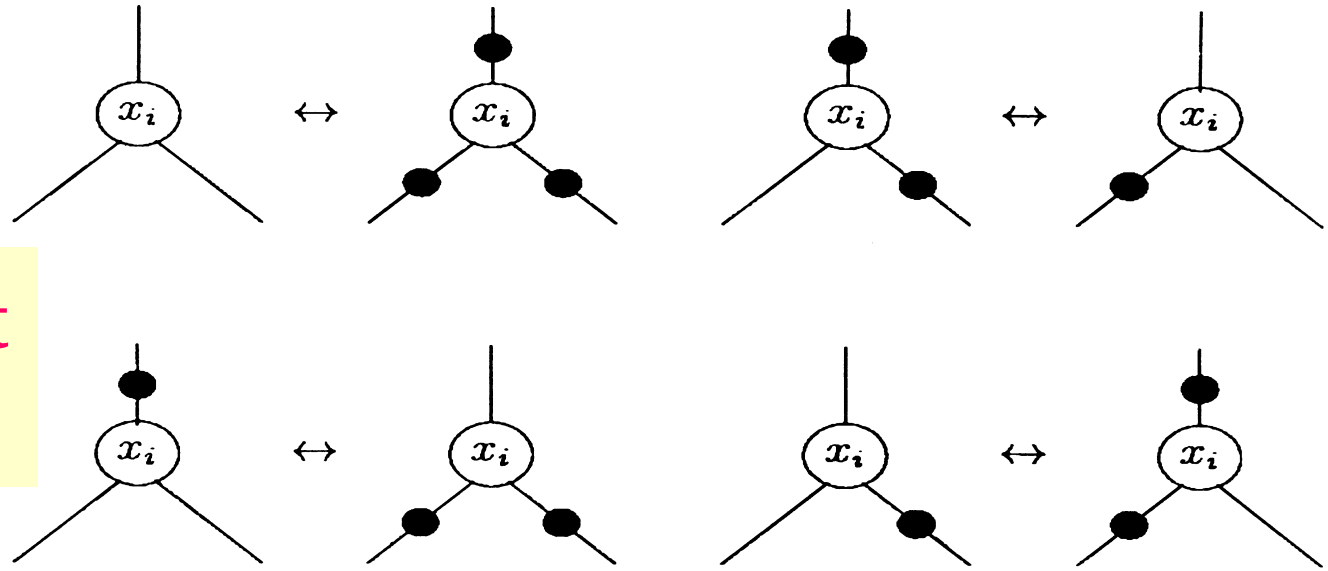


Fig. 4. Identification for canonical form for S-nodes.

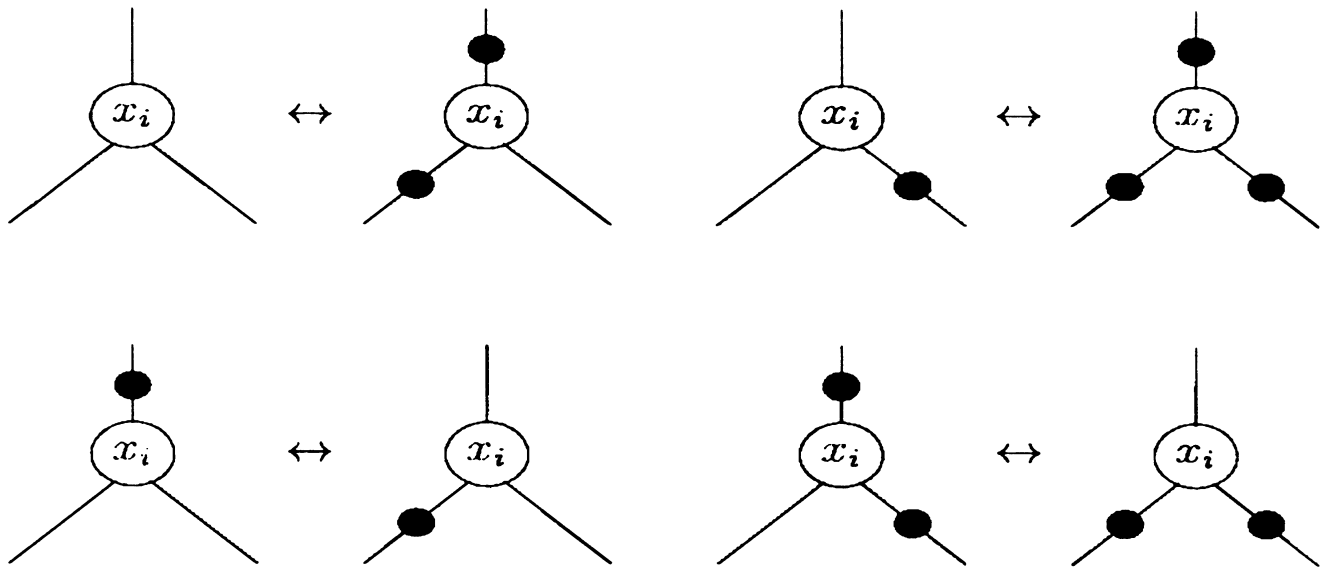


Fig. 5. Identification for canonical form for D-nodes.

XOR-operation

- D-node

$$f \oplus g = (f_0 \oplus x_i f_2) \oplus (g_0 \oplus x_i g_2) = (f_0 \oplus g_0) \oplus x_i (f_2 \oplus g_2).$$

- S-node

$$f \oplus g = \overline{x}_i (f_0 \oplus g_0) + x_i (f_1 \oplus g_1).$$

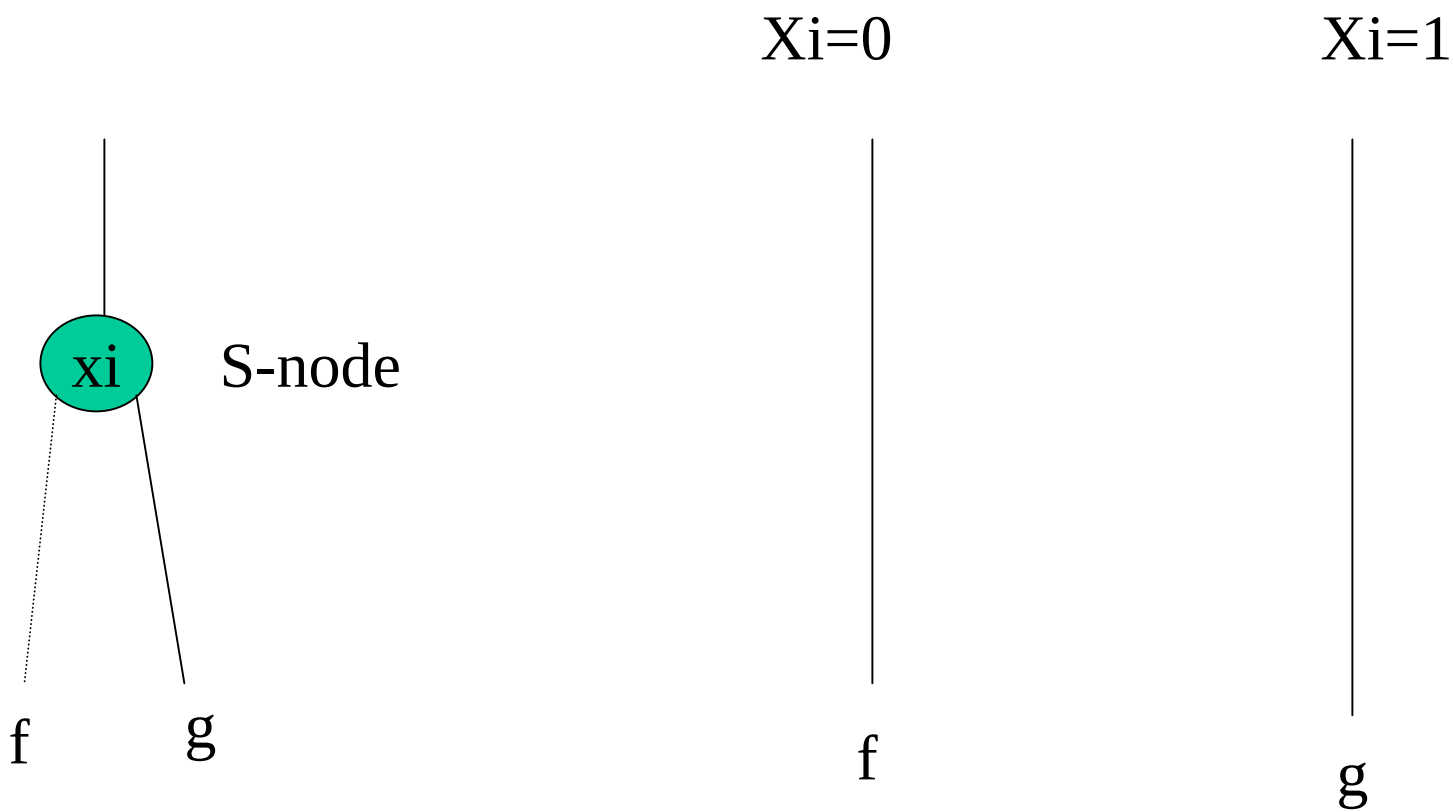
```
kfdd_xor_kfdd(F, G) {
  if (terminal case) {
    return result;
  } else if (computed-table has entry (F, G)) {
    return result;
  } else {
    let v be the top node of (F, G);
    low(v) = kfdd_xor_kfdd(F_low(v), G_low(v));
    high(v) = kfdd_xor_kfdd(F_high(v), G_high(v));
    if Shannon(v) {
    if (high(v) == low(v)) return low(v);
    } else {
    if (high(v) == 0) return low(v);
    }
    R = find_or_add_unique_table(v, low(v), high(v));
    insert_computed_table(F, G, R);
    return R;
  }
}
```

Fig. 6. Algorithm for XOR-operation.

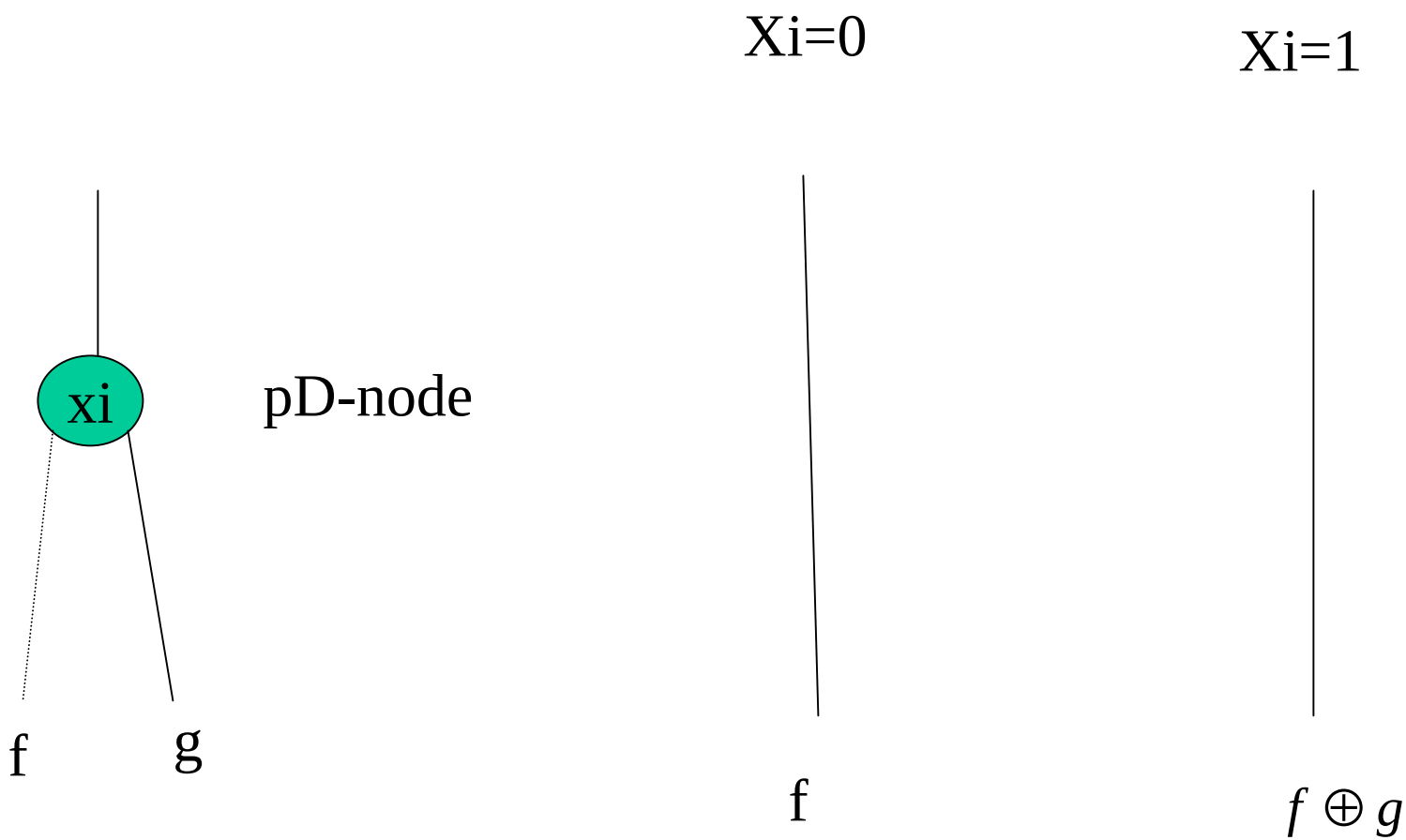
AND-operation

$$f \cdot g = (f_0 \oplus x_i f_2) \cdot (g_0 \oplus x_i g_2)$$
$$= (f_0 \cdot g_0) \oplus x_i((f_2 \cdot g_2) \oplus (f_0 \cdot g_2) \oplus (g_0 \cdot f_2)).$$

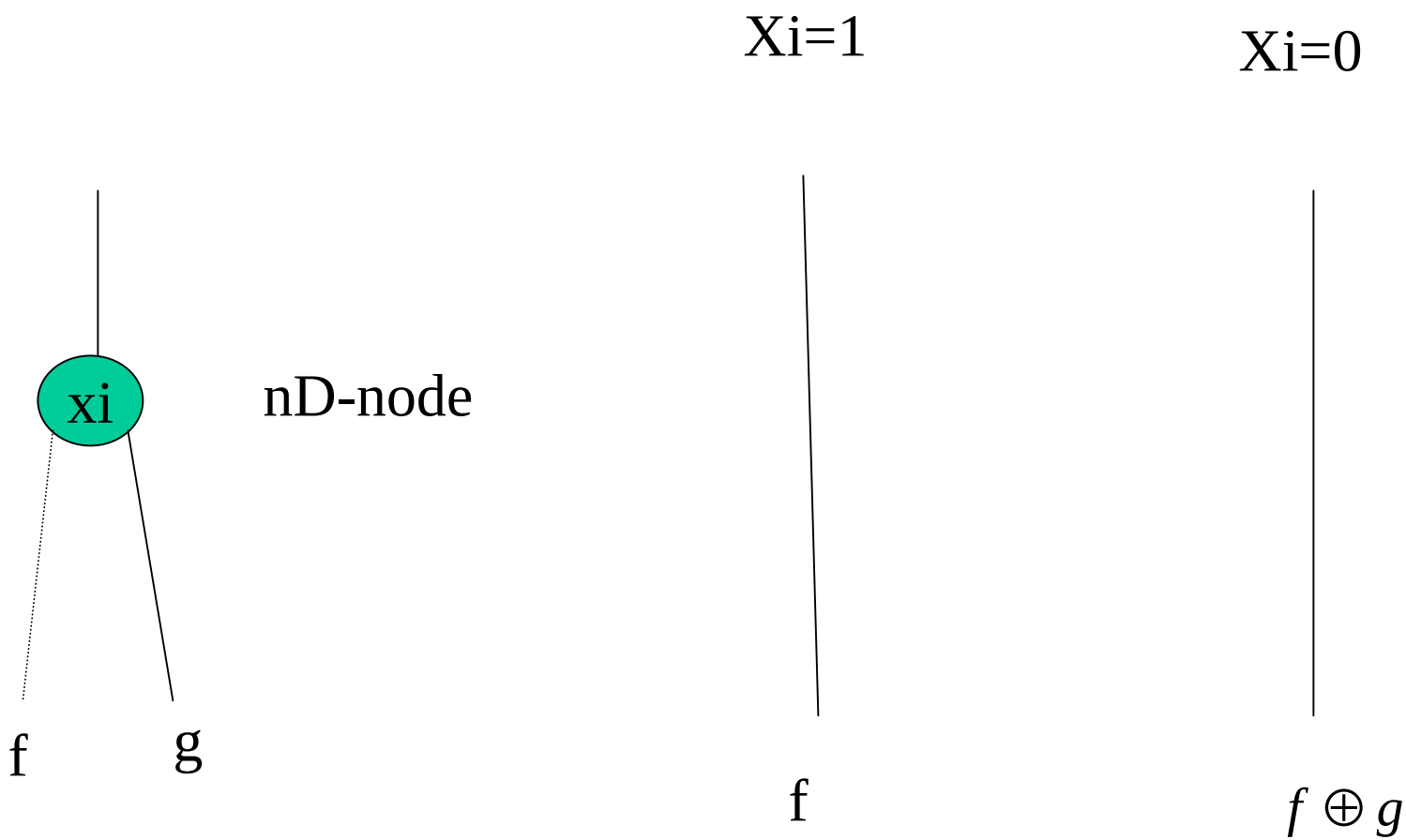
Restriction of Variables



Restriction of Variables (cond't)



Restriction of Variables (cond't)



Optimization of OKFDD-Size

- Exchange of Neighboring Variables

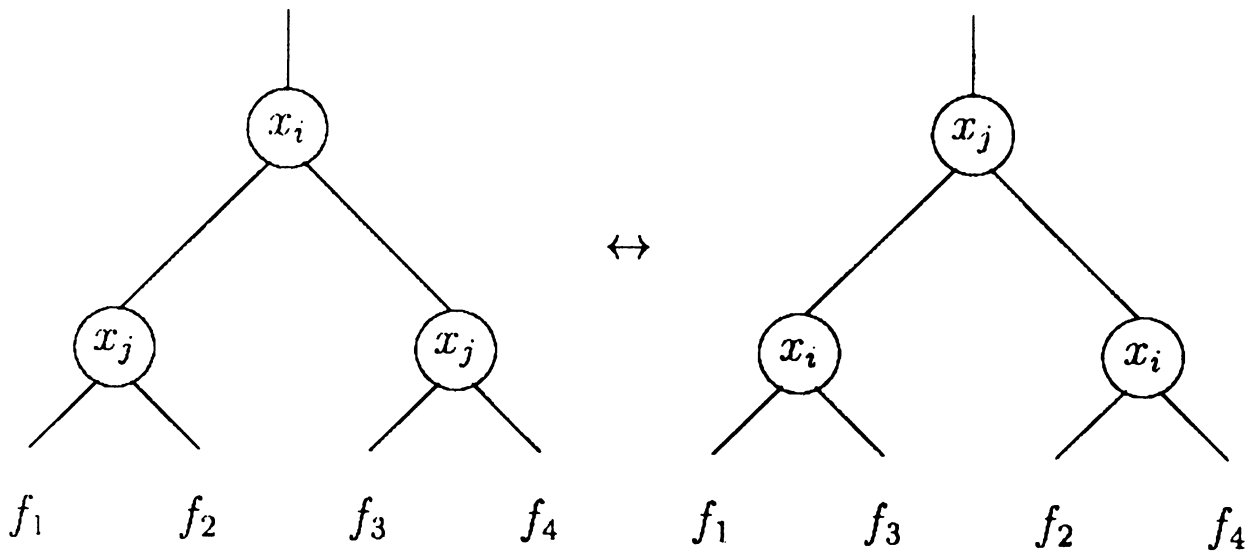


Fig. 8. Exchange of *i*th and adjacent variable.

- DTL Sifting

Experimental Results

TABLE I
COMPARISON OF SIFTED OKFDD'S AND OBDD'S

name	OKFDD-sifting			OBDD-sifting			interleaving		
	time	size	max	time	size	max	time	size	max
C432	8	1852	10000	4	1331	10000	8	31177	210000
C880	37	8907	20000	23	9118	25000	1	8654	35000
C1355	668	19217	110000	341	37960	155000	2370	40657	180000
C1908	40	5523	20000	49	9411	30000	8	17121	90000
C2670	82	5200	25000	58	6749	25000	1	26303	40000
C3540	647	57300	270000	827	59911	310000	63	153388	1100000
C5315	22	2284	10000	16	2924	20000	5	51777	145000

Experimental Results (cont'd)

TABLE II
COMPARISON OF SIFTED OKFDD'S AND OBDD'S

name	in	out	OBDD	OKFDD
C432	36	7	1.2	1.1
C499	41	32	44.8	13.4
C880	60	26	9.1	4.0
C1355	41	32	36.2	13.4
C1908	33	25	12.4	3.8
C2670	233	140	6.6	1.4
C3540	50	22	27.2	22.4
C5315	178	123	3.1	1.3
C7552	207	108	8.2	3.0
s1423	91	79	5.7	1.3
pair	173	137	4.5	1.9
rot	137	107	5.0	3.2
total			164.0	70.2

DIY PROBLEM

- Use simulated annealing algorithm for choice of a better variable ordering of the OKFDD

Simulated Annealing Basics

- Non-zero probability for "up-hill" moves.
- Probability depends on
  1. magnitude of the "up-hill" movement
  2. total search time

$$Prob(S \rightarrow S') = \begin{cases} 1 & \text{if } \Delta C \leq 0 \quad /* \text{ "}down - hill\text{" } moves */ \\ e^{-\frac{\Delta C}{T}} & \text{if } \Delta C > 0 \quad /* \text{ "}up - hill\text{" } moves */ \end{cases}$$

- $\Delta C = cost(S') - Cost(S)$
- T: Control parameter (temperature)
- Annealing schedule: $T = T_0, T_1, T_2, \ldots$, where $T_i = r^i T_0, r < 1$.

Generic Simulated Annealing Algorithm

Algorithm: Simulated_Annealing

```
1 begin
2 Get an initial solution S;
3 Get an initial temperature T > 0;
4 while not yet "frozen" do
5    for 1 ≤ i ≤ P do
6        Pick a random neighbor S' of S;
7        Δ ← cost(S') − cost(S);
         /* downhill move */
8        if Δ ≤ 0 then S ← S'
         /* uphill move */
9        if Δ > 0 then S ← S' with probability e^(−Δ/T);
10   T ← rT; /* reduce temperature */
11 return S
12 end
```

Simple solution

- Solution space

X1	X2	X3	X4

A solution

X3	X4	X1	X2

A solution

- Neighborhood structure