Binary Decision Diagrams: Theory and Applications

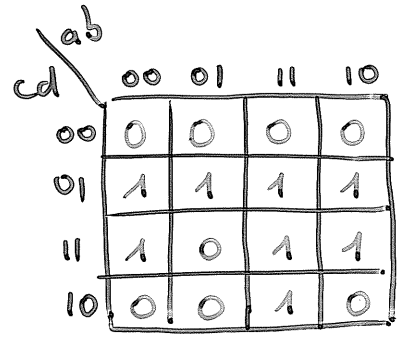
Outline:

- Representation of boolean functions
- Binary Decision Diagrams (BDDs)
- Reduced Ordered BDDs (ROBDDs)
- Basic operations with ROBDDs
- Applications
- Other types of Decision Diagrams

Representation of boolean functions

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

Truth table



Karnaugh map

Canonical sum:

$f = a'b'c'd + a'b'cd + a'bc'd + ab'c'd + ab'cd + abcd' + abcd$

Canonical product:

$f = (a+b+c+d)(a+b+c'+d)(a+b'+c+d)(a+b'+c'+d)(a+b'+c'+d')(a'+b+c+d)(a'+b+c'+d)(a'+b'+c+d)$

Sum of products:

$f = abc + b'd + c'd$

Product of sums:

$f = (c+d)(b+d)(a+b'+c')$

Multi-level form:

$x = bc$

$f = ax + x'd$

Canonical representation

A form is canonical if the representation of a function in that form is unique.

Interest for canonical forms: Equivalence test

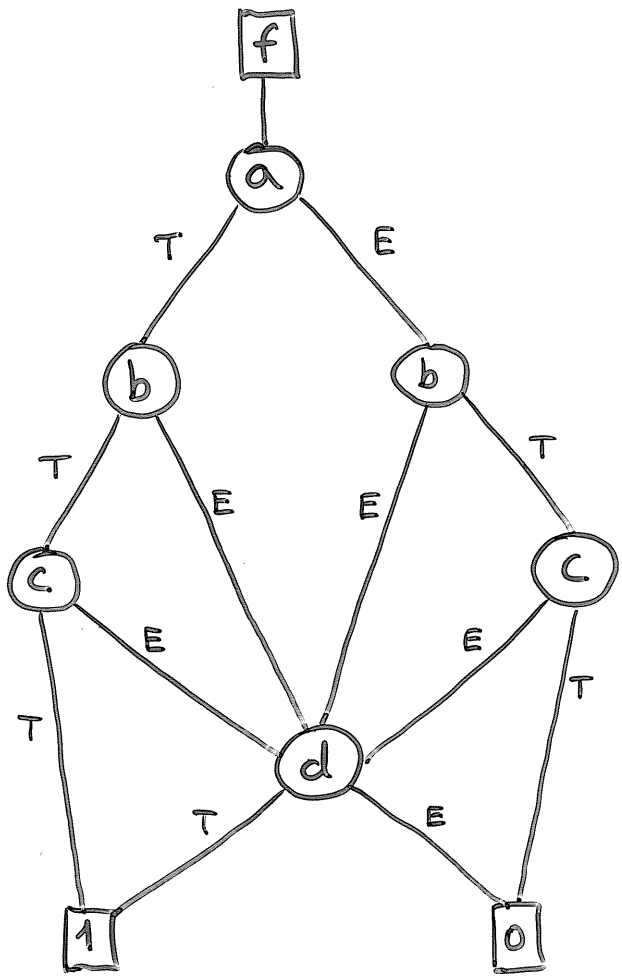
$f_1 \equiv f_2 \iff$ their representation in a canonical form is the same

Problem: Representations in canonical forms are large (tipically exponential in the number of variables)

BDDs (Binary Decision Diagrams)

- Canonical form
- Compact for many functions

BDD: an example

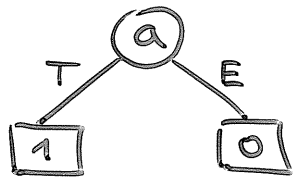


This BDD represents the function

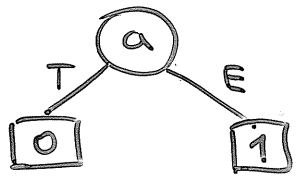
$$f = abc + b'd + c'd$$

What is the value of $f(1,0,1,0)$?

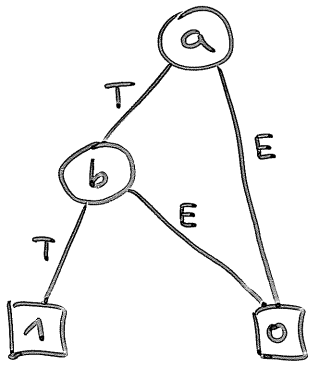
ELEMENTARY FUNCTIONS



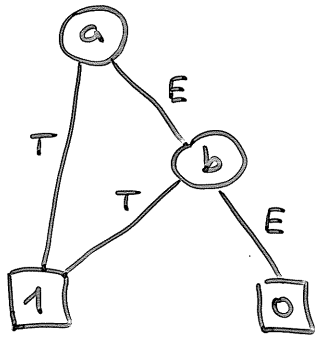
a



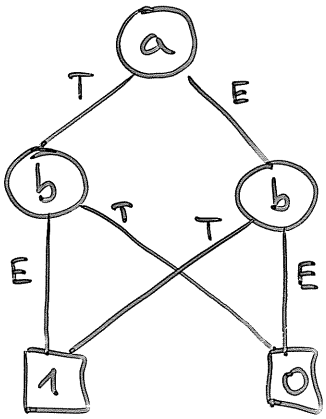
$\bar{a}$



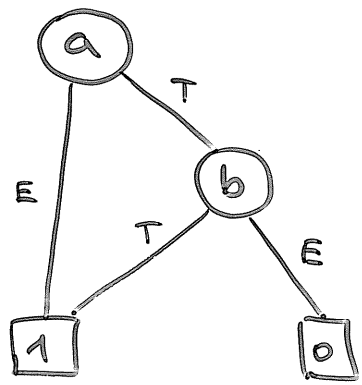
$a\,b$



$a + b$



$a \oplus b$



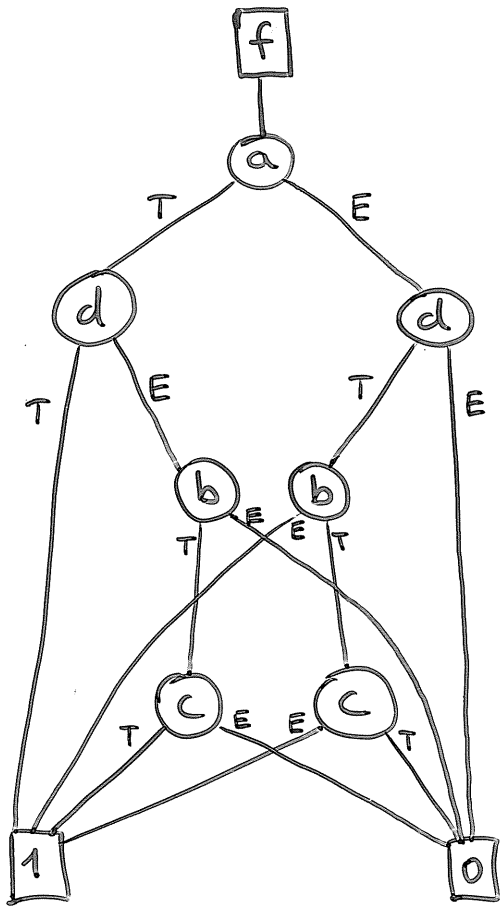
$a \Rightarrow b$

$(\bar{a} + b)$

Variable ordering

Variable ordering of the previous BDD: $a \leq b \leq c \leq d$

If we take $a \leq d \leq b \leq c$ we have



Ordered BDD:

Variables appear in the same order along all paths from the root to the leaves

Building a BDD

Example: $f = abc + b'd + c'd$

Ordering: $b \leq c \leq d \leq a$

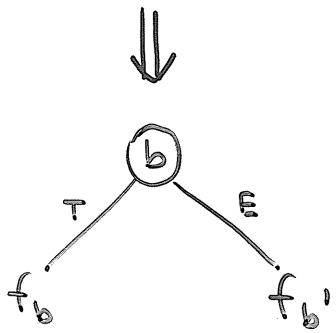
Theoretical basis: Boole's expansion
(or Shannon expansion)

$$f = x_i f_{x_i} + x_i' f_{x_i'}$$

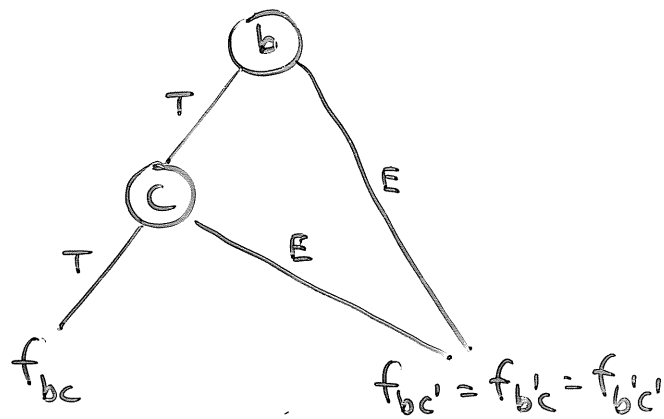
where

$$f_{x_i}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$
$$f_{x_i'}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

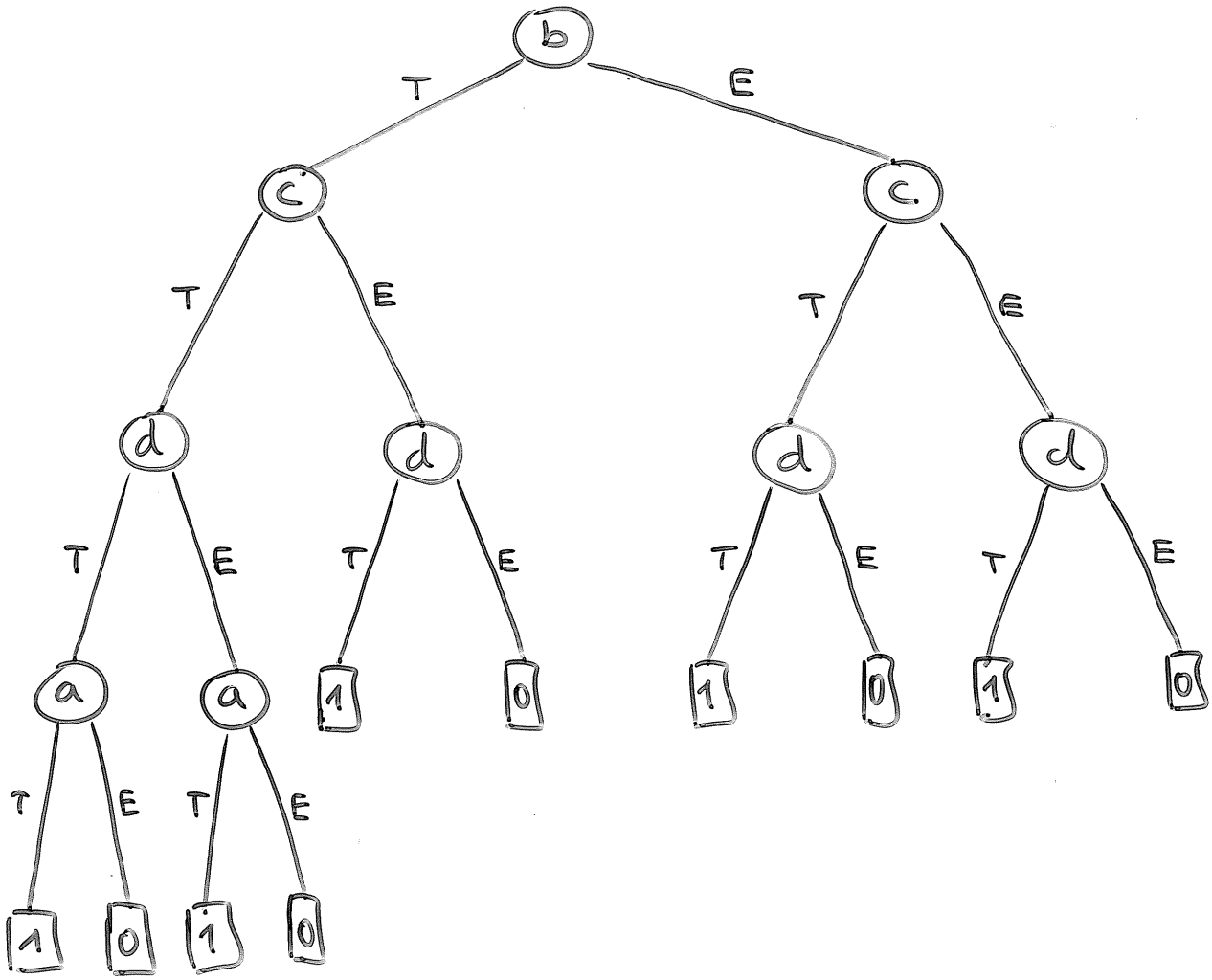
$f_b = ac + c'd$ $\qquad$ $f_{b'} = d$



$f_{bc} = a$
$f_{bc'} = d$
$f_{b'c} = d$
$f_{b'c'} = d$

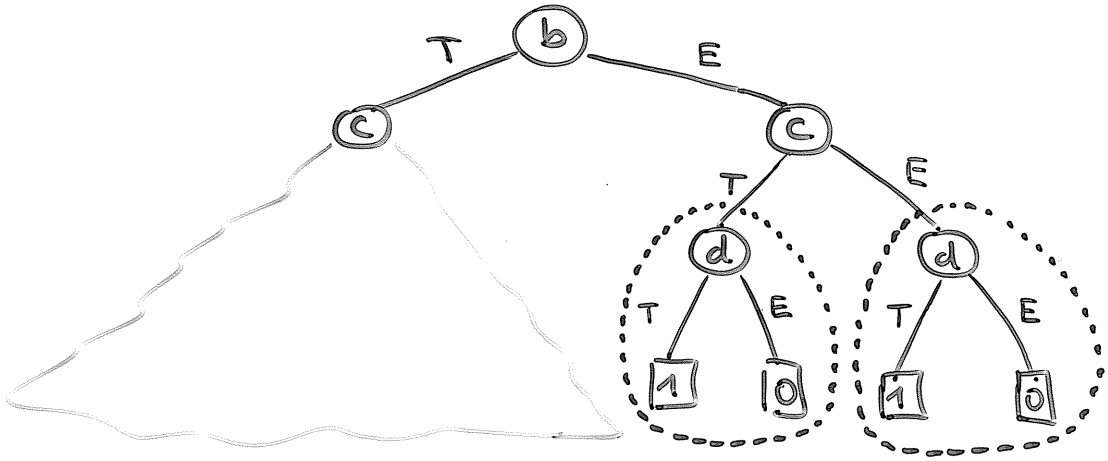


without sharing cofactors ...

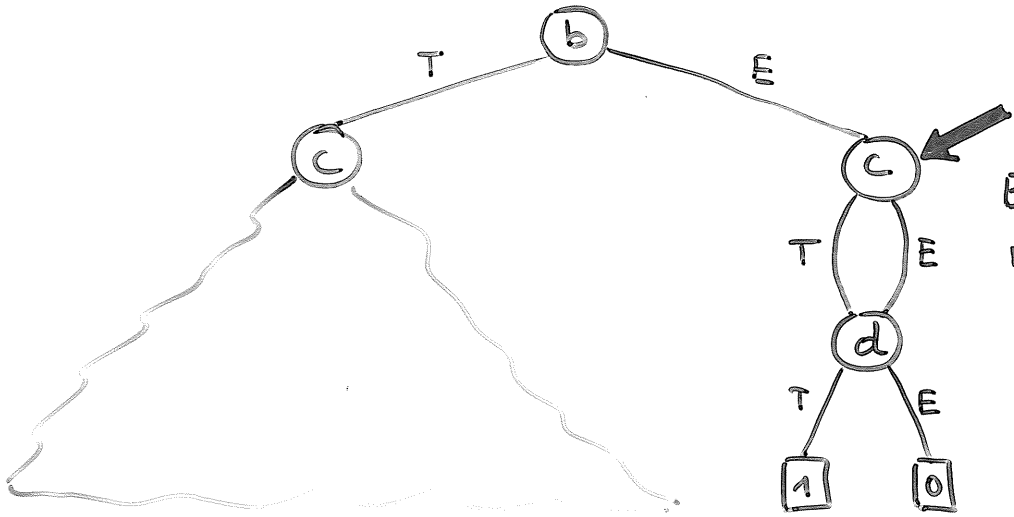


This BDD is not reduced !

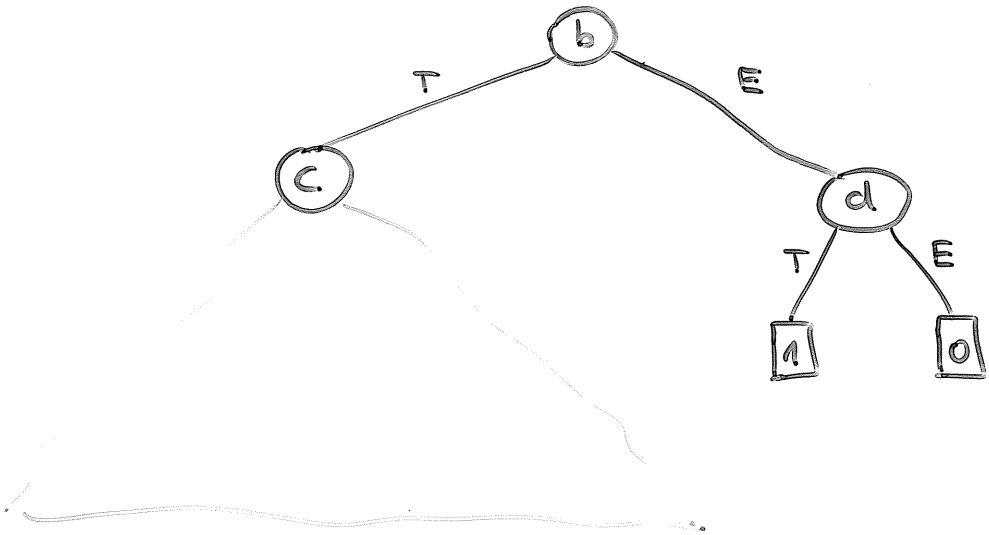
REDUCTIONS



Isomorfic subgraphs



Elimination of redundant nodes



ROBDDs

Given an OBDD, by iteratively applying the reductions:

- Identification of isomorfic subgraphs
- Elimination of redundant nodes

a <u>reduced</u> OBDD is obtained (ROBDD)

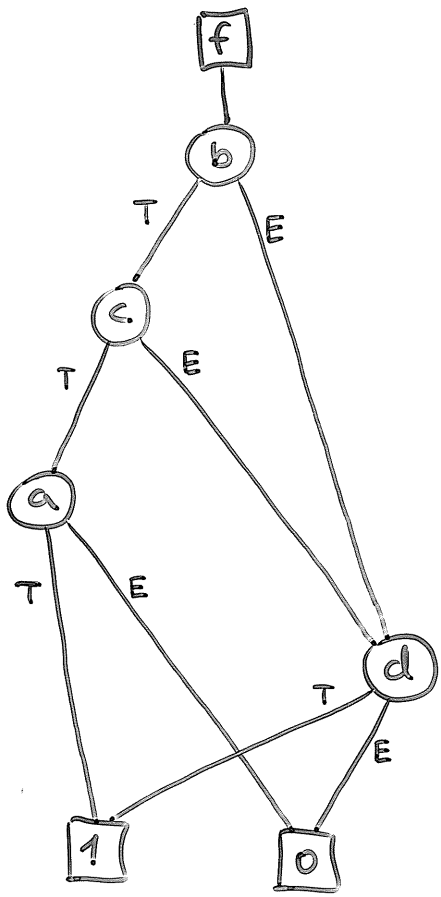
> A Reduced Ordered BDD is a canonical form.

Characteristics of ROBDDs:

- The size of the BDD is exponential in the number of variables in the worst case. However BDDs are a compact representation for many functions
- The logical AND and OR of BDDs have polynomial complexity in the size of the BDDs
- Satisfability and tautology can be solved in constant time
- Covering problems can be solved in linear time

Optimal ordering for the example:

$b \trianglelefteq c \trianglelefteq a \trianglelefteq d$



- The size and shape of a BDD depend on the variable ordering
- Each node of a BDD represents a variable

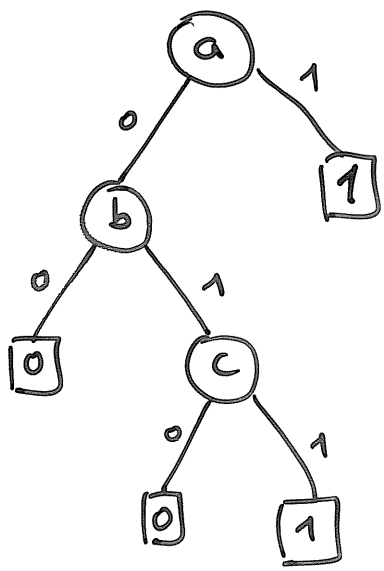
Finding the best variable order is NP complete

Heuristics:

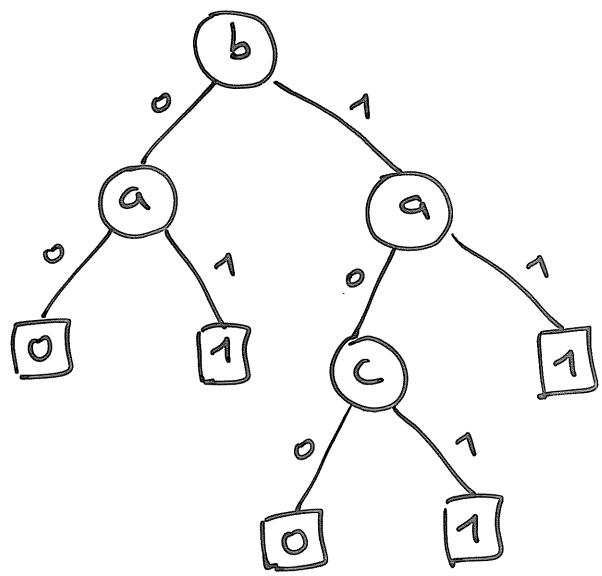
- Variables with close relation should be near
- Variables with high control should be at higher positions

Example: $f = a + bc$

$a < b < c$



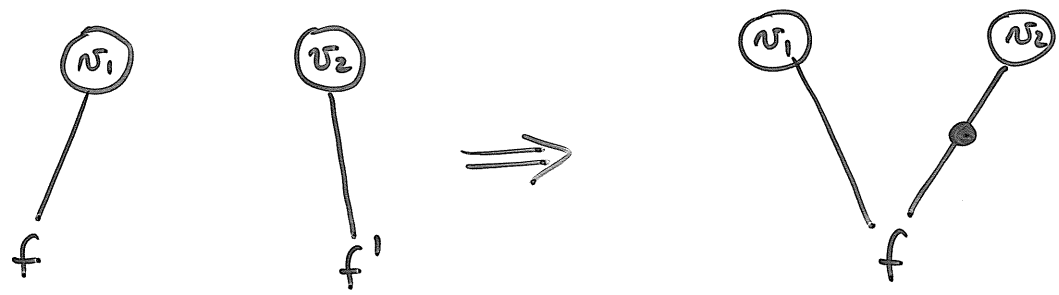
$b < a < c$



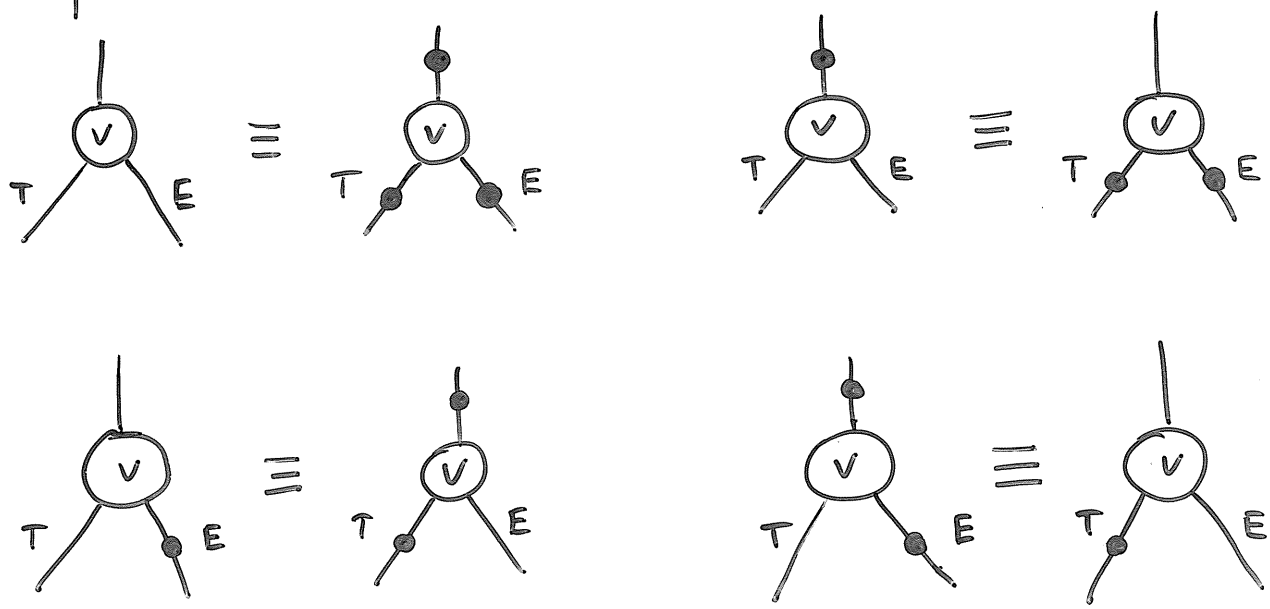
Complement Arcs

Two BDDs, on representing f and the other representing f', have the same structure, but with the value of the leaves interchanged

Complement arc:



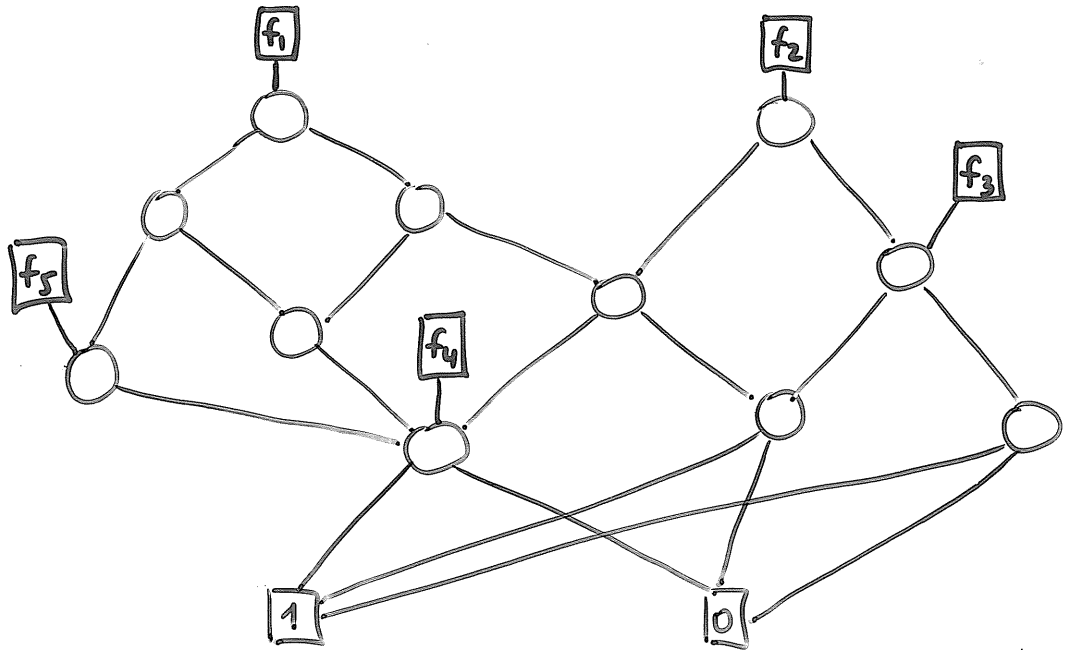
To maintain canonicity, the constrain that only "ELSE" arcs can be complemented is imposed.



Implementation of a BDD Package

- Shared BDDs

  Data structure : Multi-rooted DAG



- Unique Table

  Dictionary of the functions (nodes) represented in the DAG. Each time a new function is created, its existence in the "Unique Table" is checked. BDDs must never be reduced. The "Unique Table" is implemented as a hash table

- Strong canonicity

  Checking for the equivalence of two functions just requires checking that the pointers in the DAG are identical

- Computed Table

  To improve the efficiency of BDD operations, a table containing the most recently computations is created, in order to avoid re-computation. (cache of the most recently results)

- Memory management

  garbage collection of all nodes not used

Operations with BDDs

Building the BDD for a function usually starts by building the BDDs of the variables of the function. Next, more complex functions are built by means of binary operations.

$$f \ \underline{op} \ g$$

Boole's expansion:

$$f \ \underline{op} \ g = x_i \left(f_{x_i} \ \underline{op} \ g_{x_i} \right) + x_i' \left(f_{x_i'} \ \underline{op} \ g_{x_i'} \right)$$

The ITE (if-then-else) operator

$$ite\ (F, G, H) = F \cdot G + F' \cdot H$$

where F, G, and H are boolean functions

All 2-argument boolean operators can be expressed in terms of "ite"

Operator	Expression	Equivalent form
0	0	0
AND (F, G)	$F \cdot G$	ite (F, G, 0)
$F > G$	$F \cdot G'$	ite (F, G', 0)
F	F	F
$F < G$	$F' \cdot G$	ite (F, 0, G)
G	G	G
XOR (F, G)	$F \oplus G$	ite (F, G', G)
OR (F, G)	$F + G$	ite (F, 1, G)
NOR (F, G)	$(F + G)'$	ite (F, 0, G')
XNOR (F, G)	$(F \oplus G)'$	ite (F, G, G')
NOT (G)	G'	ite (G, 0, 1)
$F \geq G$	$F + G'$	ite (F, 1, G')
NOT (F)	F'	ite (F, 0, 1)
$F \leq G$	$F' + G$	ite (F, G, 1)
NAND (F, G)	$(F \cdot G)'$	ite (F, G', 1)
1	1	1

ITE algorithm

$$Z = ite(F, G, H)$$

Let "v" be the top variable of F, G, and H

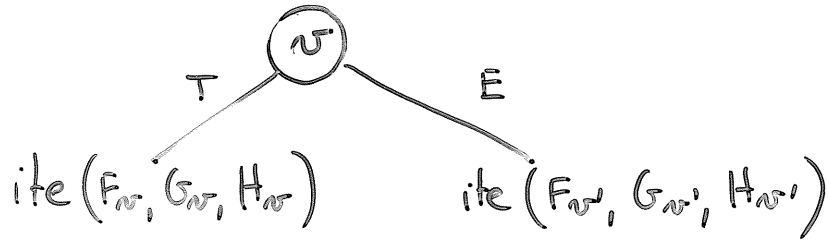
$$Z = v\, Z_v + v'\, Z_{v'}$$

$$= v\,(FG + F'H)_v + v'\,(FG + F'H)_{v'}$$

$$= v\,(F_v G_v + F'_v H_v) + v'\,(F_{v'} G_{v'} + F'_{v'} H_{v'})$$

$$= ite(v,\ ite(F_v, G_v, H_v),\ ite(F_{v'}, G_{v'}, H_{v'}))$$

$$= (v,\ ite(F_v, G_v, H_v),\ ite(F_{v'}, G_{v'}, H_{v'}))$$



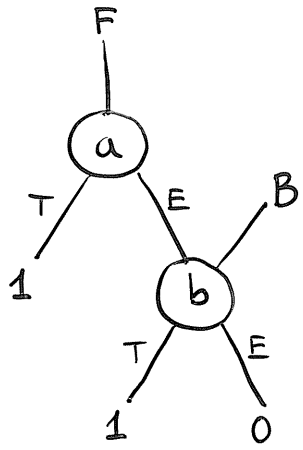
Terminal cases:

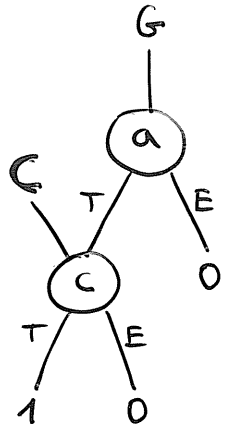
$ite(1, F, G) = ite(0, G, F) = ite(F, 1, 0) = ite(G, F, F) = F$

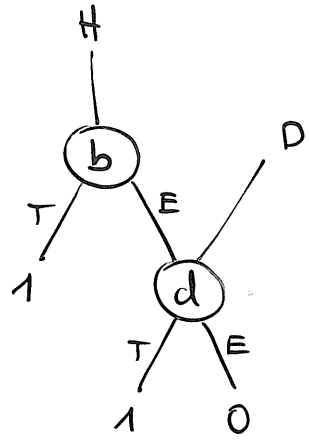
The ITE Algorithm (cont)

```
ite (F, G, H) {
    if (terminal case) {
        return result;
    } else if (computed-table has entry {F, G, H}) {
        return result;
    } else {
        let v be the top variable of {F, G, H}
        T = ite (F_v, G_v, H_v);
        E = ite (F_v', G_v', H_v');
        if T equals E return T;
        R = find_or_add_unique_table (v, T, E);
        insert_computed_table ({F, G, H}, R);
        return R;
    }
}
```

Example

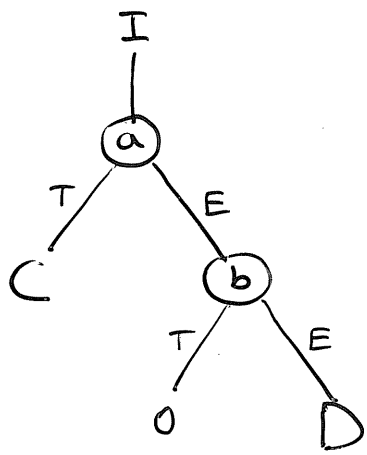






$F = a + b$ $\qquad$ $G = ac$ $\qquad$ $H = b + d$

$$I = ite(F, G, H)$$
$$= (a, ite(F_a, G_a, H_a), ite(F_{a'}, G_{a'}, H_{a'}))$$
$$= (a, ite(1, C, H), ite(B, 0, H))$$
$$= (a, C, (b, ite(B_b, 0_b, H_b), ite(B_{b'}, 0_{b'}, H_{b'})))$$
$$= (a, C, (b, ite(1, 0, 1), ite(0, 0, D)))$$
$$= (a, C, (b, 0, D))$$



Abstractions

$$\exists x f = f_x + f_{\bar{x}}$$

$$\forall x f = f_x \cdot f_{\bar{x}}$$

Example: $f = xy + \bar{x}z + wy$

$$\exists x f = y + z$$

$$\forall x f = y(w+z)$$

```
Eabs (f, x) {   /* x is a cube */
    if(terminal case) return result;
    if (computed table has entry {f,x})
        return result;
    v = top var (f);
    if (v = topvar (x)) {
        x' = x_v;
        T = Eabs (f_v, x'); E = Eabs (f_v̄, x');
        R = OR (T, E)
    } else {
        T = Eabs (f_v, x); E = Eabs (f_v̄, x);
        R = ite (v, T, E);
    }
    insert computed table ({f, x, R});
    return R;
}
```

Solving Satisfiability Problems with BDDs

- Finding one satisfying assignment

  With complement arcs there is only one leave node

Finding a variable assignment such that $f = 1$ is solved by finding a path in the BDD with an even number of complement arcs

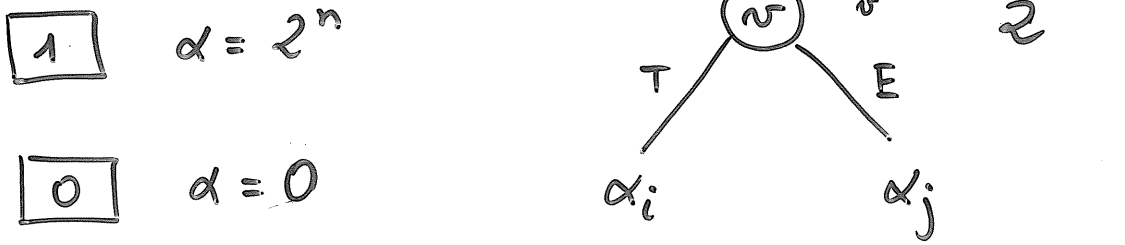
```
OneSat (v, p, sat) {
    if (v is terminal node) return p;
    sat [v → index] = 1;
    if (OneSat (v → T, p, sat)) return 1;
    sat [v → index] = 0;
    if (v → E is complemented) complement p;
    return OneSat (v → E, p, sat);
}
```

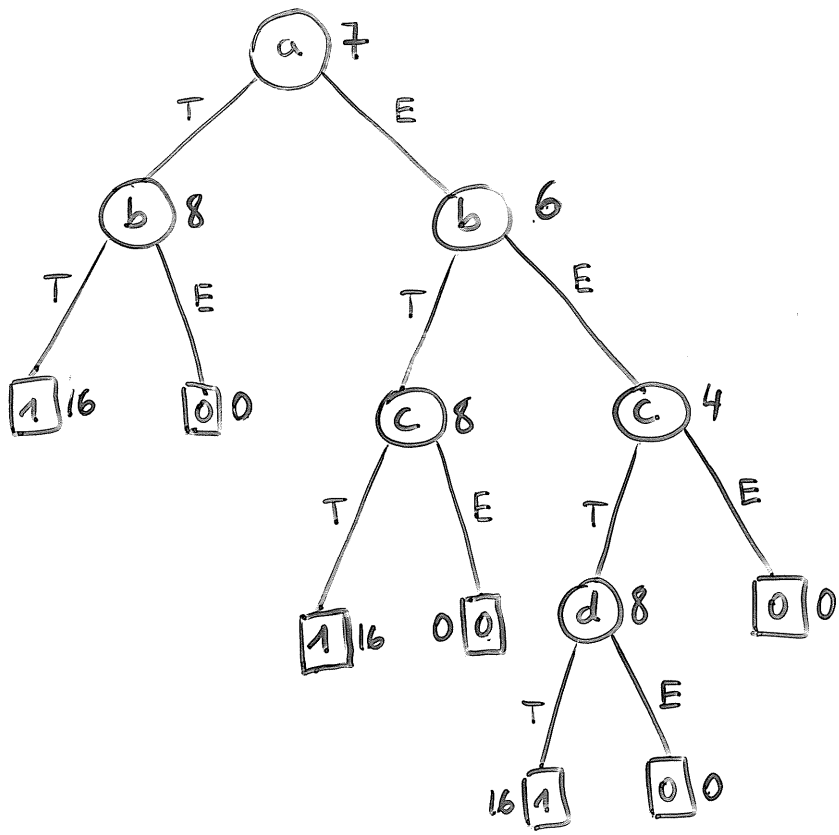
Complexity $O(n)$

At most $2n + 1$ nodes are visited

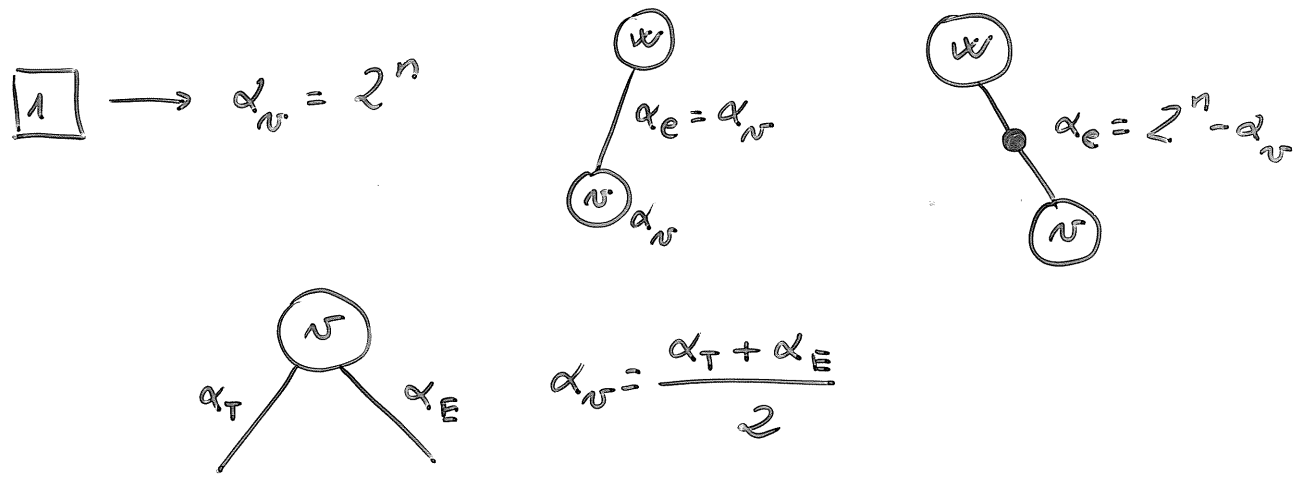
• Counting the number of satisfying assignments

- Let us assume that we have n variables
- For each node "i" we define "α_i" as:





If complement arcs are used:




```
SatHowMany (v, n) {
    if (v is terminal node) return 2^n;
    if (v is in table) return result from table;
    countT = SatHowMany (v → T, n);
    countE = SatHowMany (v → E, n);
    if (v → E is complemented) countE = 2^n - countE;
    count = (countT + countE) / 2;
    insert (v, count) in table;
    return count;
}
```

Complexity: $O(2^n)$ without table

$O(n)$ with table

• Finding one minimum-cost satisfying assignment

- Each variable x_i has an associated cost c_i
- The cost of an assignment is the sum of the costs of the selected variables (a variable x_i is selected when $x_i = 1$)

- Definition:
  - The length of an E arc is 0
  - The length of a T arc out of a node labeled x_i is c_i

Theorem:

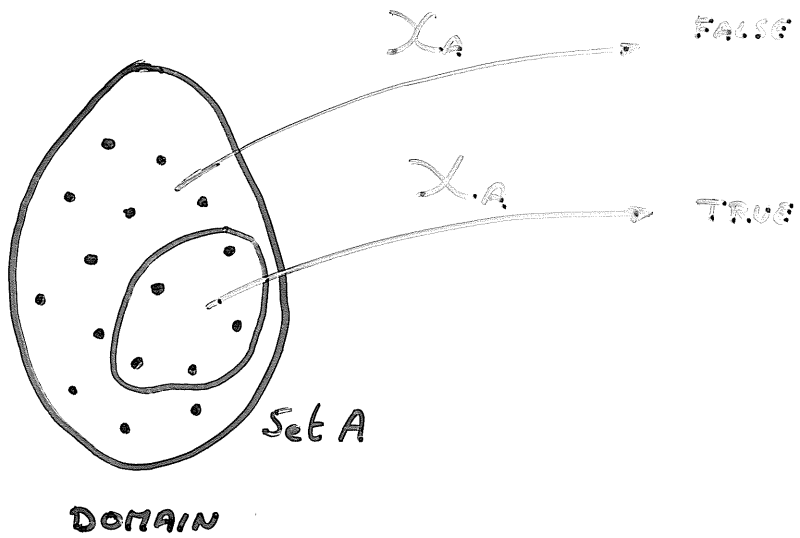
Let F be a BDD for $f(x_1, \ldots, x_m)$. Then the minimum cost assignment to $x_1, \ldots, x_m$ that is a solution to

$$f(x_1, \ldots, x_m) = 1$$

is given by the shortest path connecting the root of F to the terminal node

Characteristic Functions

Identifies sets of elements in a given domain:



Each element in a domain encoded with a binary vector in B^n.

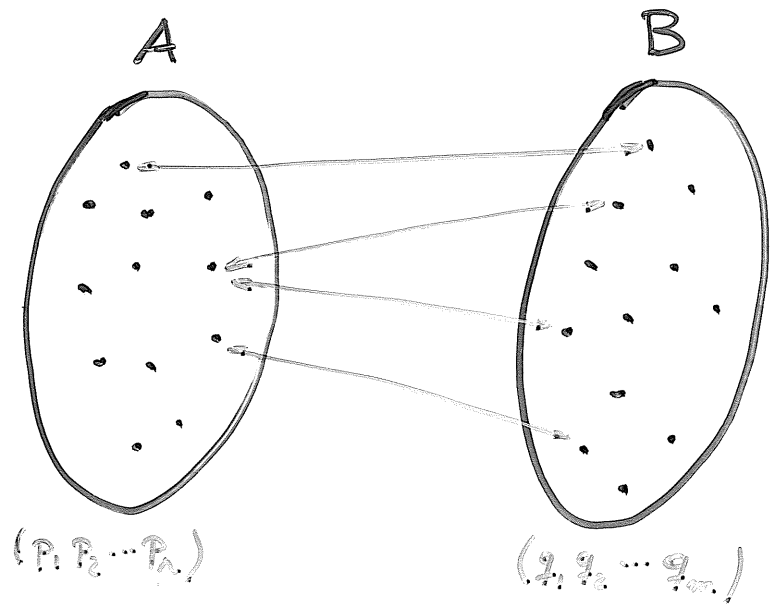
$$\chi_A = \sum_{\forall a_i \in A} \varepsilon(a_i)$$

Logic operations $\equiv$ Set operations.

$$A \cup B \equiv \chi_A + \chi_B$$
$$A \cdot B \equiv \chi_A \cdot \chi_B$$
$$\overline{A} \equiv \overline{\chi_A} \cdot \chi_{DOMAIN}$$

Set Relations

Characteristic Functions allow the creation of Relations between Sets.



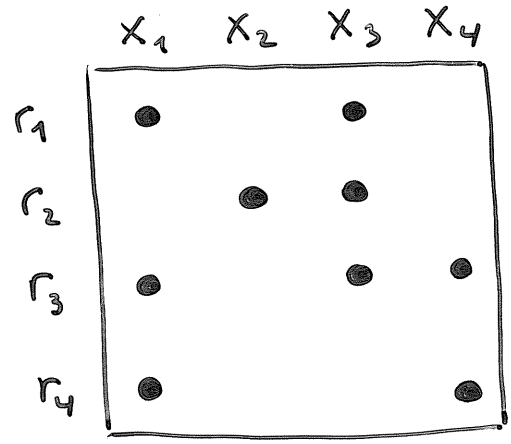
Relation between A and B : $R(A, B)$

- $a \in A,\ b \in B \quad (a, b) \in R(A, B)$

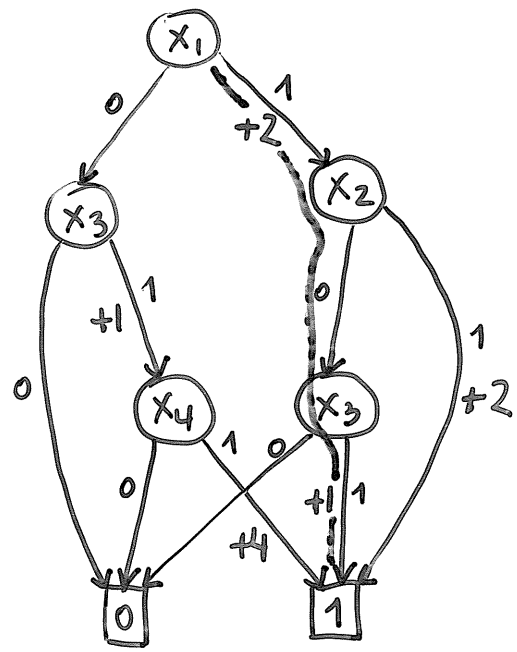
Characteristic Function:

$$\chi_R(A, B) = 1 \Leftrightarrow \exists_{a \in A,\, b \in B} : (a, b) \in R$$

Covering problems, 0-1 Integer Linear Programming



$$\min \quad 2x_1 + 2x_2 + x_3 + 4x_4$$

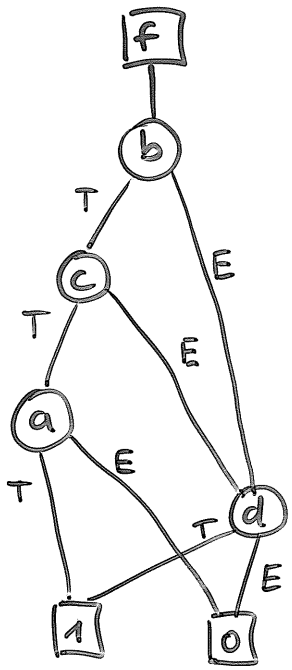


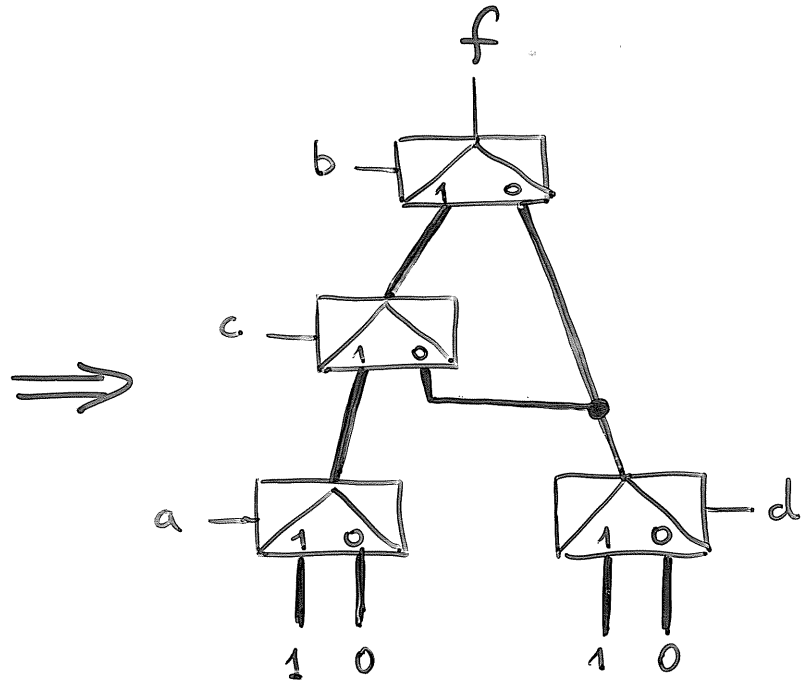
Shortest path: $x_1 \to \bar{x}_2 \to x_3 \to 1$

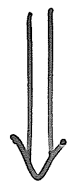
Solution: $x_1 = x_3 = 1 \quad x_2 = x_4 = 0$

Application: Logic synthesis

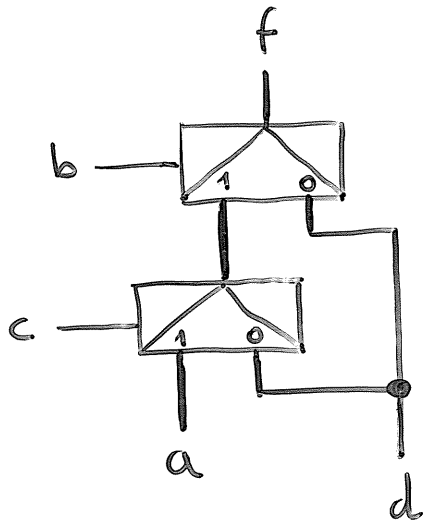
$$f = abc + b'd + c'd$$



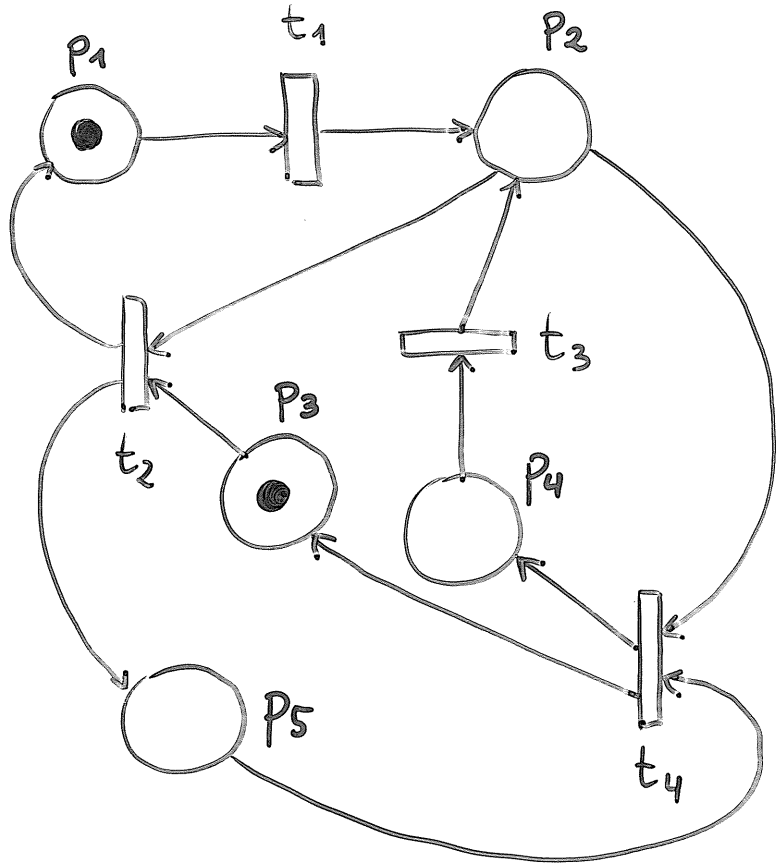




After simplifying logic



Analysis of Petri Nets



Each marking corresponds to a minterm of B^5

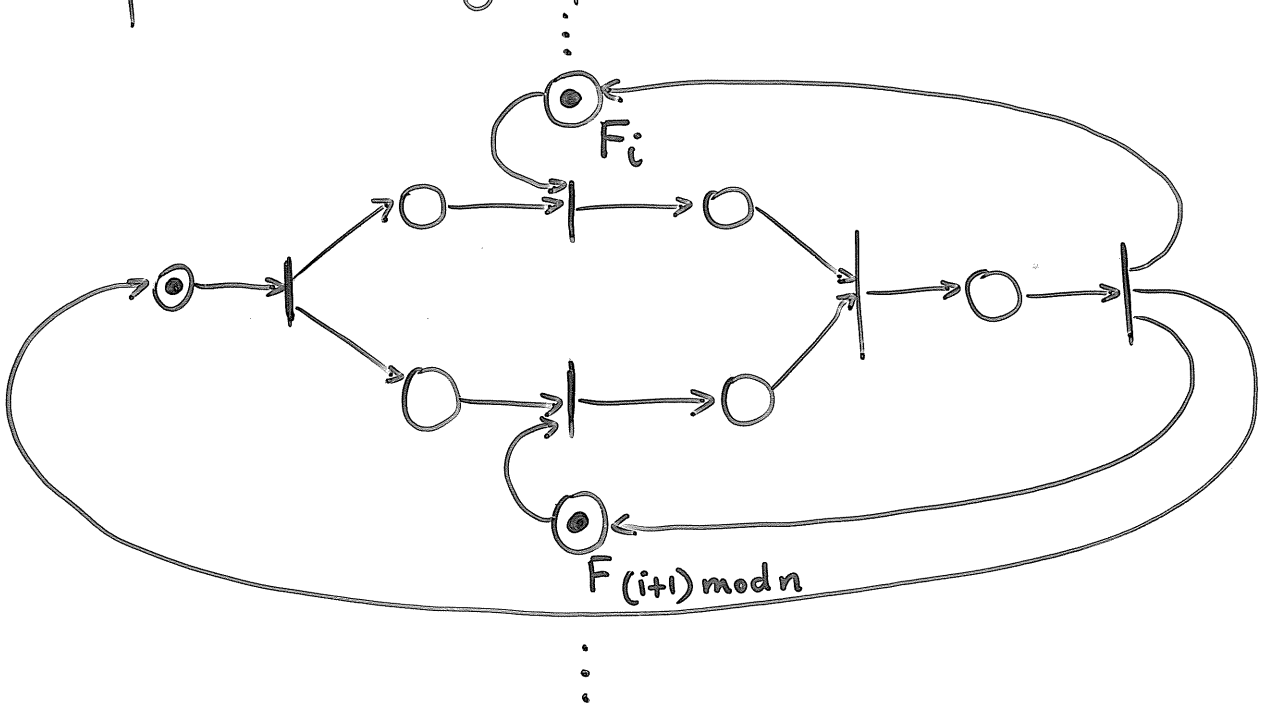
Initial marking: $P_1 P_2' P_3 P_4' P_5'$

Set of marking

$$M = \{\{P_2, P_5\}, \{P_2, P_3, P_5\}, \{P_1, P_2, P_5\}, \{P_1, P_2, P_3, P_5\}, \{P_1, P_2, P_3, P_4, P_5\}\}$$

$$X_M = P_1 P_2 P_3 P_5 + P_2 P_4' P_5$$

Example: dinning philosophers

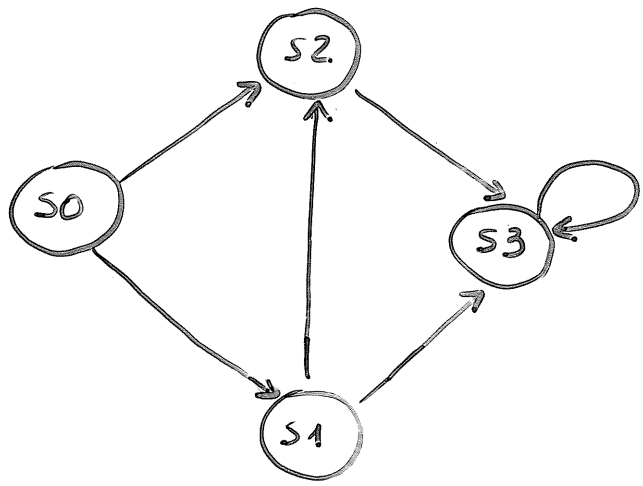


n philosophers ⟶ 7n places
5n transitions

Problem: deadlock detection

# phil.	# markings	BDD size	# iters	CPU (secs.)
4	466	183	3	3
8	2.2×10^5	431	17	66
12	1.0×10^8	679	25	438
16	4.7×10^{10}	927	33	1918
20	2.2×10^{13}	1175	41	6360

Graph representation



Binary encoding

S0 → 00
S1 → 01
S2 → 10
S3 → 11

Adjacency matrix

$$A = \begin{array}{c|cccc} ab \backslash cd & 00 & 01 & 10 & 11 \\ \hline 00 & 0 & 1 & 1 & 0 \\ 01 & 0 & 0 & 1 & 1 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 0 & 1 \end{array}$$

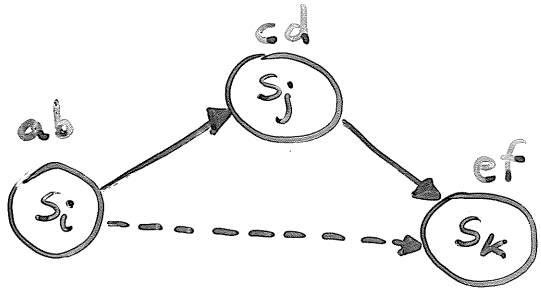
Characteristic function

$$A = a'b'c'd + a'b'cd' + a'bcd' + a'bcd + ab'cd + abcd$$

$$A = a'b'(c \oplus d) + a'bc + acd$$

Transitive closure

$$A^n = A^{n-1} \cdot A$$



$$abcd \cdot cdef = abcdef$$

$$A \cdot A = (a'b'c'd + a'b'cd' + a'bc + acd) \cdot (c'd'e'f + c'd'ef' + c'de + cef) =$$

$$= a'b'c'de + a'b'cd'ef + a'bcef + acdef$$

$\exists$ Abstraction of S_j (cd)

$$\exists_{cd} A \cdot A = a'b'e + a'b'ef + a'bef + aef =$$

$$= a'b'e + ef$$

Variable renaming

$$a'b'c + cd$$

$$A^2 = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ 00 & 0 & 0 & 1 & 1 \\ 01 & 0 & 0 & 0 & 1 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 0 & 1 \end{array}$$

$$T = A \vee A^2 = a'b'(c \oplus d) + a'bc + acd + a'b'c + cd$$

$$T = a'cd' + a'b'd + cd$$

$$T = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 0 & 1 & 1 & 1 \\ 01 & 0 & 0 & 1 & 1 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 0 & 1 \end{array}$$

Is S0 reachable from S3 ?

$$R_{S3 \to S0} \equiv \frac{ab}{S3}\,\frac{c'd'}{S0} \qquad R_{S3 \to S0} \subseteq T \; ?$$

$$\overline{R_{S3 \to S0}} + T = 1 \; ?$$

$$a' + b' + c + d + a'cd' + a'b'd + cd = a' + b' + c + d \neq 1$$

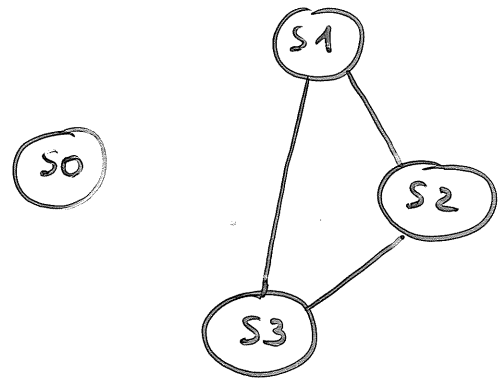
Is S3 reachable from S0 ?

$$\overline{R_{S0 \to S3}} + T = 1 \; ?$$

$$a + b + c' + d' + a'cd' + a'b'd + cd = \underline{1}$$

Connected components

$$A = \begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$



$$A = a'b'c'd' + (a+b)(c+d)$$

1.- Extract one minterm: i.e. $ab'c'd$

2.- $\exists_{cd} \longrightarrow ab'$

3.- All nodes connected to ab'

$$A \cdot (ab') = ab'c + ab'd$$

4.- $\exists_{ab} \longrightarrow c+d \xrightarrow{\text{Renaming}} a+b$

$\underline{a+b}$ is the charact. function of one connected component

5.- Subtract $(a+b)$ from the graph

$$A \cdot (a+b)' = a'b'c'd' = B$$

6.- B contains all rows s.t. $(a+b)'$

$$\vdots$$

iterate $\Rightarrow$ Components $\begin{cases} a+b & \text{3 nodes} \\ a'b' & \text{1 node} \end{cases}$

Other types of DDs

EXOR-based expansions

$$f = \overline{x} f_0 \oplus x f_1 = \overline{x} f_0 + x f_1 \quad \text{(Shannon)}$$

$$f = f_0 \oplus x f_2 \quad \text{(positive Davio expansion)}$$

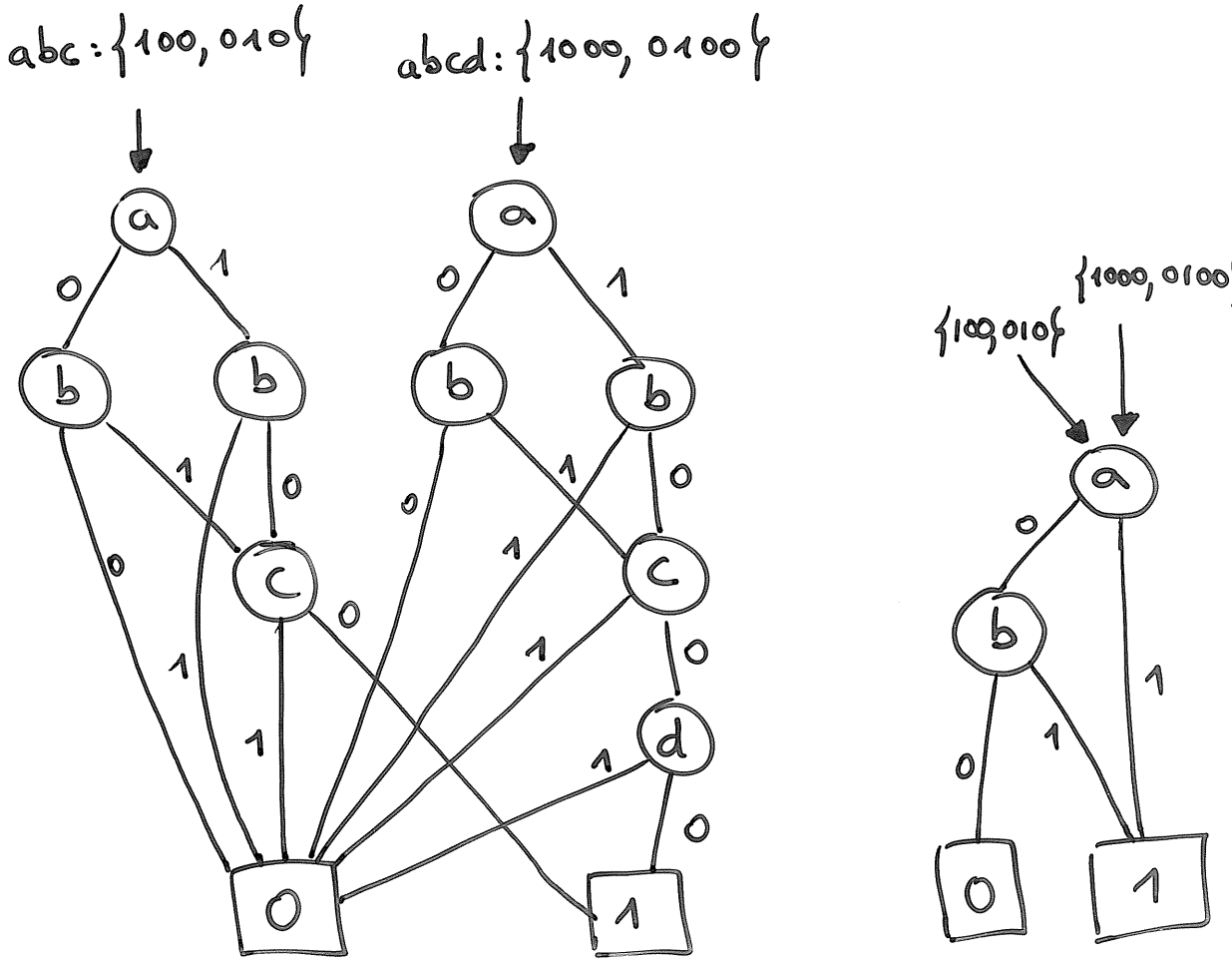
$$f = f_1 \oplus \overline{x} f_2 \quad \text{(negative Davio expansion)}$$

with $f_2 = f_0 \oplus f_1$

- Different expansions can be combined in the same DD.
- Appropriate for arithmetic logic

Zero-supressed BDDs (ZBDDs)

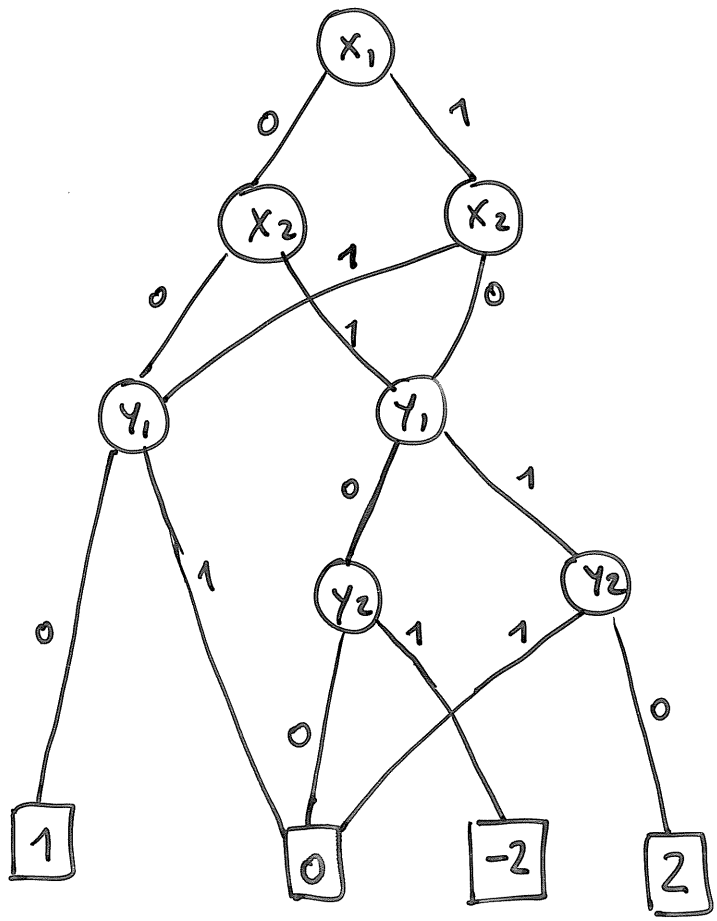
- Eliminate all nodes with the 1-edge pointing to [0]
- Appropriate for sparse encoding functions



Multi-Terminal BDDs (MTBDDs)

- Can represent multi-valued functions

x_1x_2 \ y_1y_2	00	01	10	11
00	1	1	0	0
01	0	-2	2	0
10	0	-2	2	0
11	1	1	0	0



and many other types...

- Edge - Valued BDDs
  - Representation of linear expressions
- Binary Moment Diagrams
  - Paths represent product terms of algebraic expressions
- Multi - valued Decision Diagrams
  - More than 2 branches for each node

.
.
.

Conclusions

→ BDDs : compact representation for many boolean functions

→ Isomorfism between Boolean Algebras

- Minterms of a boolean function
- Arcs of a graph
- Markings of a Petri net
- ... and many others

→ We can apply "boolean reasoning" (BDDs) to solve problems in different domains

→ Efficient packages to manage BDDs

Bibliography:

- S.B. Akers
  "Binary decision diagrams"
  IEEE Trans. on Computers, June 1978

- R.E. Bryant
  "Graph-based algorithms for Boolean function manipulation"
  IEEE Trans. on Computers, Aug. 1986

- T. Sasao and M. Fujita (editors)
  "Representations of Discrete Functions"
  Kluwer Academic Publishers, 1996

- S. Minato
  "Binary Decision Diagrams and Applications for VLSI CAD"
  Kluwer Academic Publishers, 1996

- R.E. Bryant
  "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams"
  ACM Computing Surveys, Sept. 1992

- F.M. Brown
  "Boolean Reasoning: The Logic of Boolean Equations"
  Kluwer Academic Publishers, 1990