

# BoolTool: A Tool for Manipulation of Boolean Functions

Petr Fišer, David Toman

Czech Technical University in Prague  
Dept. of Computer Science and Engineering  
fiserp@fel.cvut.cz, tomand1@fel.cvut.cz



# Outline

- Motivation
- Basic description of BoolTool
- Network representation & manipulation
- Ternary tree & its minimization
- Experimental results
- Conclusions

# Basics & Motivation

BoolTool is a tool for manipulation of Boolean functions (Boolean networks)

Why?

- No simple tools performing elementary Boolean operations upon logic functions (or Boolean networks), like AND, OR, XOR, ..., are not available
- Command-line tool is required, to enable scripting

# BoolTool

BoolTool is a tool for manipulation of Boolean functions  
(Boolean networks)

➤ Operations supported:

- Basic Boolean operations (NOT, AND, OR, XOR, ...)
- Computation of a complement of a Boolean function
- Transformation of a Boolean network into an AND-OR-NOT representation
- Transformation of a Boolean network into a network consisted of 2-input NAND or NOR gates only
- Transformation of a Boolean network into a CNF or DNF representation, thus, collapsing the multi-level network to obtain a two-level representation
- Satisfiability (SAT) solving
- Cofactor computation

# BoolTool

BoolTool is a tool for manipulation of Boolean functions  
(Boolean networks)

➤ Interface:

- Source file: PLA (2-level) or structural VHDL (multi-level), CNF (DIMACS)
- Individual PLA outputs are considered as independent functions
- Destination: PLA, VHDL, BLIF, CNF
- Operation description: in a VHDL-like style, or interactively

# BoolTool

## Example of the usage

### input.vhdl

```
x <= a or (b and not c);  
y <= b or c;
```

BoolTool input.vhdl output.vhdl operation.vhdl

### output.vhdl

```
z <= ( a or ( b and ( not c ) ) ) xor ( b or c );
```

### operation.vhdl

```
z <= x xor y;
```

# BoolTool

## Example of the usage

**input.vhdl**

```
x <= a or (b and not c);  
y <= b or c;
```

a	b	c	
			1
1	1	1	1

a	b	c	
	1	1	1
	1	1	1

**operation.vhdl**

```
z <= x xor y;
```

BoolTool input.vhdl output.pla operation.vhdl

**output.pla**

```
.i 3  
.o 1  
.ilb a b c  
.ob z  
0-1 1  
100 1
```

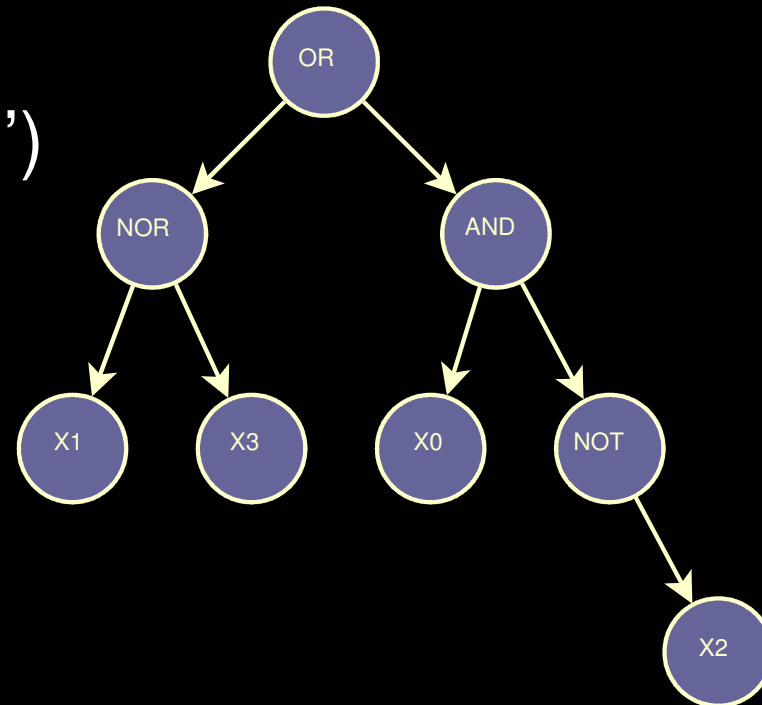
a	b	c	
	1	1	
1			

# Internal Network Representation

One binary tree for each function

Internal nodes are operations (binary, unary),  
leaves are variables

$(x_1 \text{ nor } x_3) \text{ or } (x_0 \text{ and } x_2')$



# Network Manipulation

## > Simple Boolean operations upon two trees

- The new operator just becomes a root, the trees are its children.

## > Complementation of the tree (NOT)

- By recursively traversing the tree from root to leaves. DeMorgan's laws may be used, to produce an AND-OR-NOT tree only.

## > Network collapsing

- The tree is recursively converted into AND-OR-NOT tree. Then negations are propagated to leaves.

## > SAT solving

- The tree is recursively converted into AND-OR-NOT tree. Then '1' value is propagated to leaves. Conflicting variables are easily identified during recursion.
- All the solutions to the SAT are found by this way

# Ternary Tree - Motivation

- Collapsing the tree into a two-level representation (PLA) is extremely time and memory consuming
- Duplicate terms are often produced during the collapsing procedure
- Terms that can be absorbed by already produced terms are often produced

Proof: try yourself manually. I.e.  $(ab+ac+bc)'$

# Ternary Tree - Motivation

An efficient structure to represent a set of terms is needed.

## Requirements:

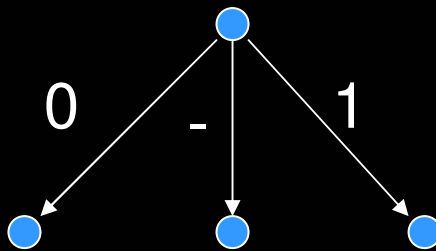
- Duplicities are avoided
- Large number of stored terms is expected (up to millions)
- Short insertion time
- Low memory demands
- Fast and on-line minimization

# Ternary Tree

Used as a **storage of terms**.

It is not a function representation (like BDDs)!

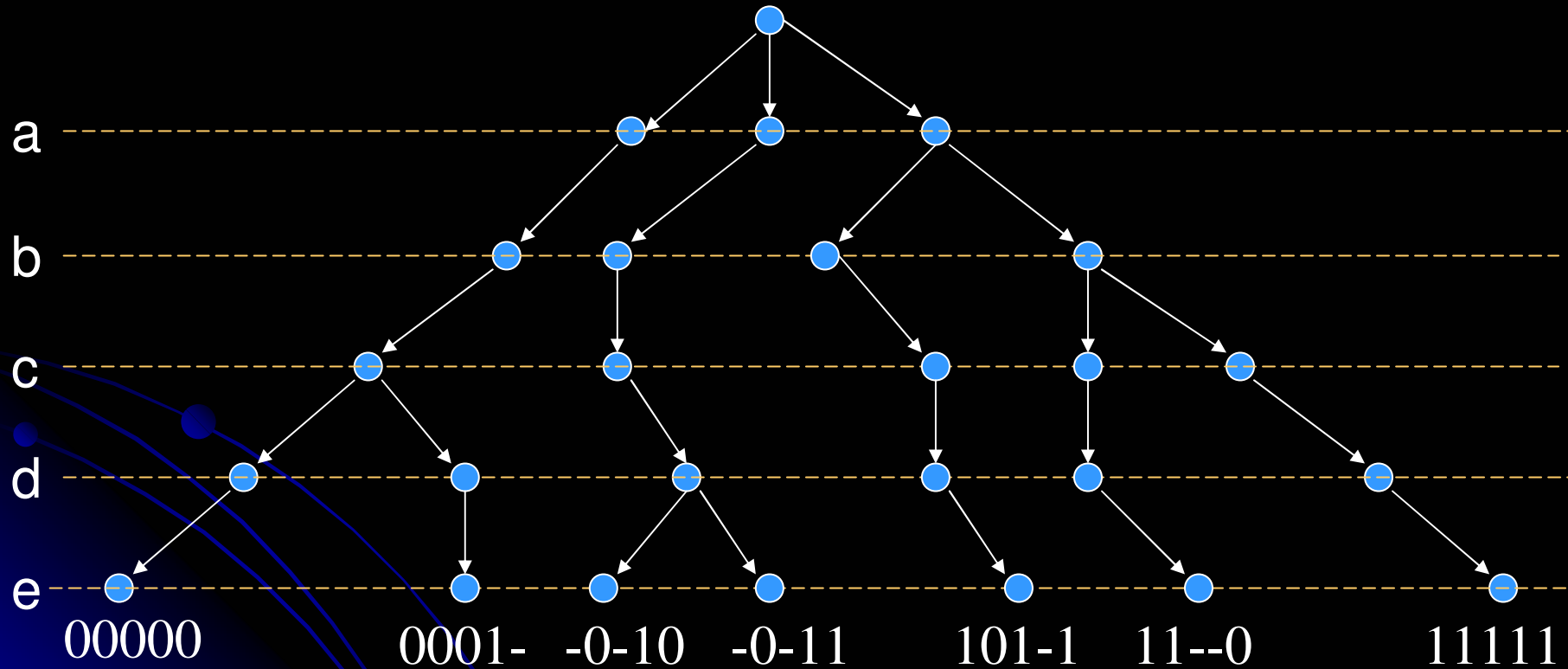
Ternary tree node:



Depth = number of input variables

# Ternary Tree

$$a'b'c'd'e' + a'b'c'd + b'de' + b'de + ab'ce + abe' + abcde$$



# Ternary Tree

## Properties:

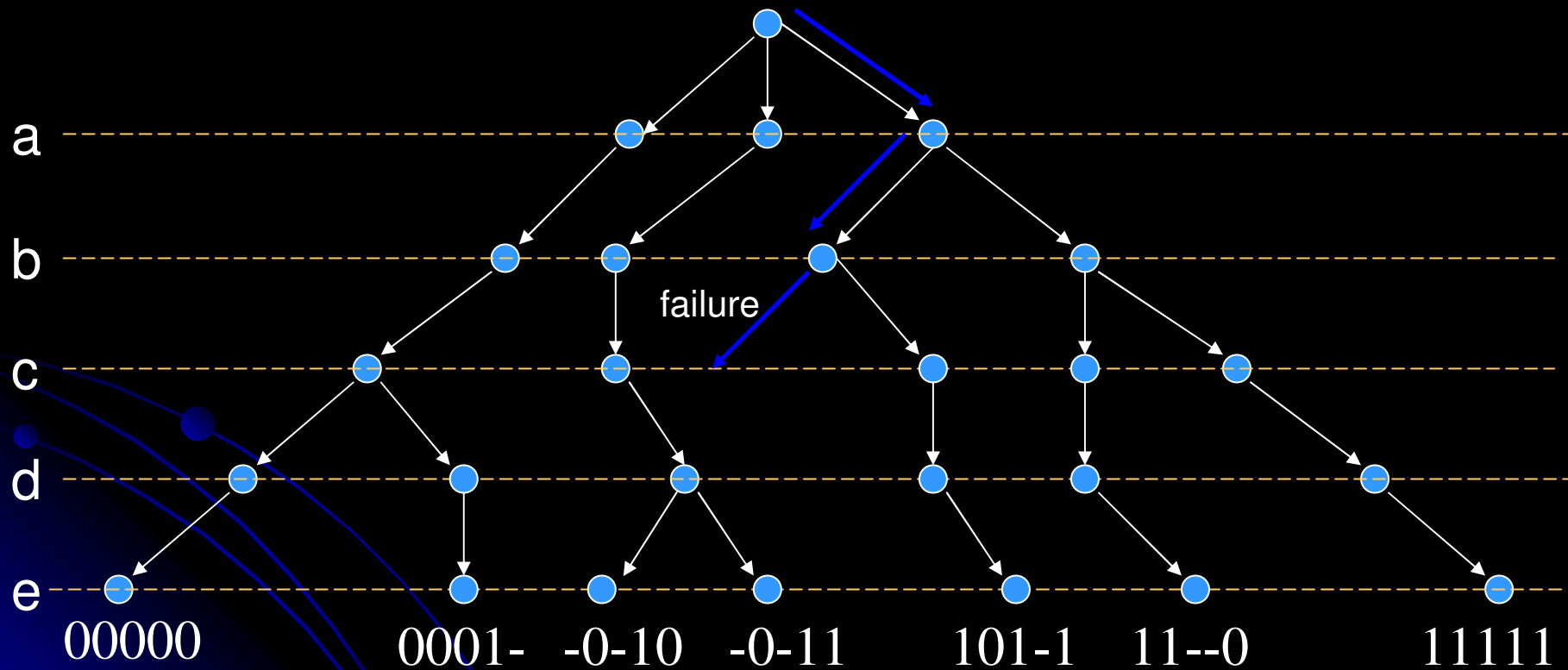
- Insertion of a term in  $O(n)$
- Looking up a term in  $O(n)$
- Size: between  $n$  and  $3^{n+1}$

## Comparison of look-up speeds:

	On success	On failure
Ternary tree	$n$	1 ... $(n-1)$
Truth table	$n \dots n.p$	$n.p$

# Ternary Tree Look-up Example

Searching: ab'c'd (1001-)



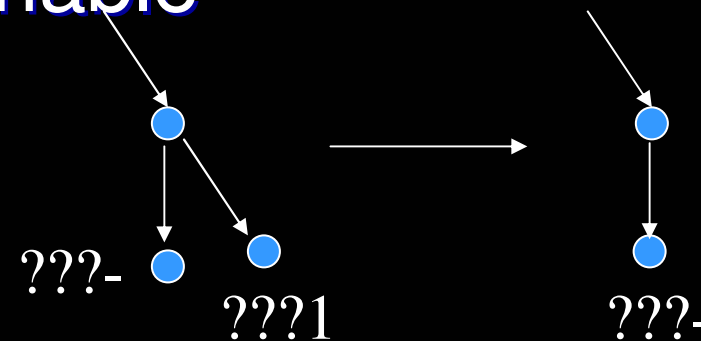
# TT Minimization

Very simple. Only two rules applicable to the leaves only:

## ➤ Absorption of one variable

$$a + ab = a$$

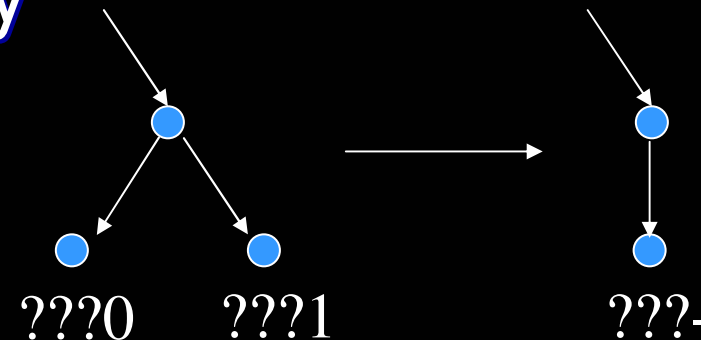
$$(00-) + (001) = (00-)$$



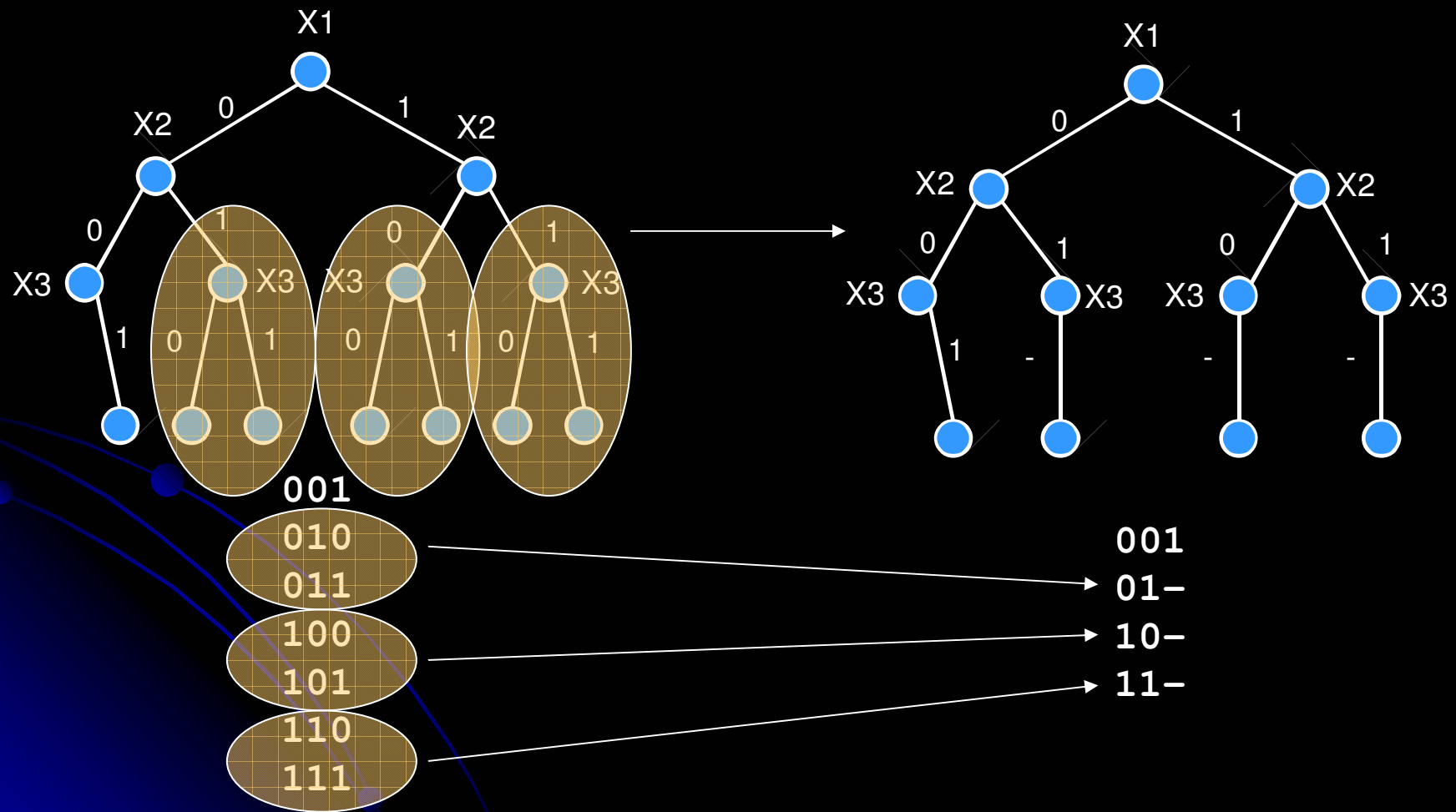
## ➤ Complement property

$$ab + a'b = b$$

$$(000) + (001) = (00-)$$



# TT Minimization Example

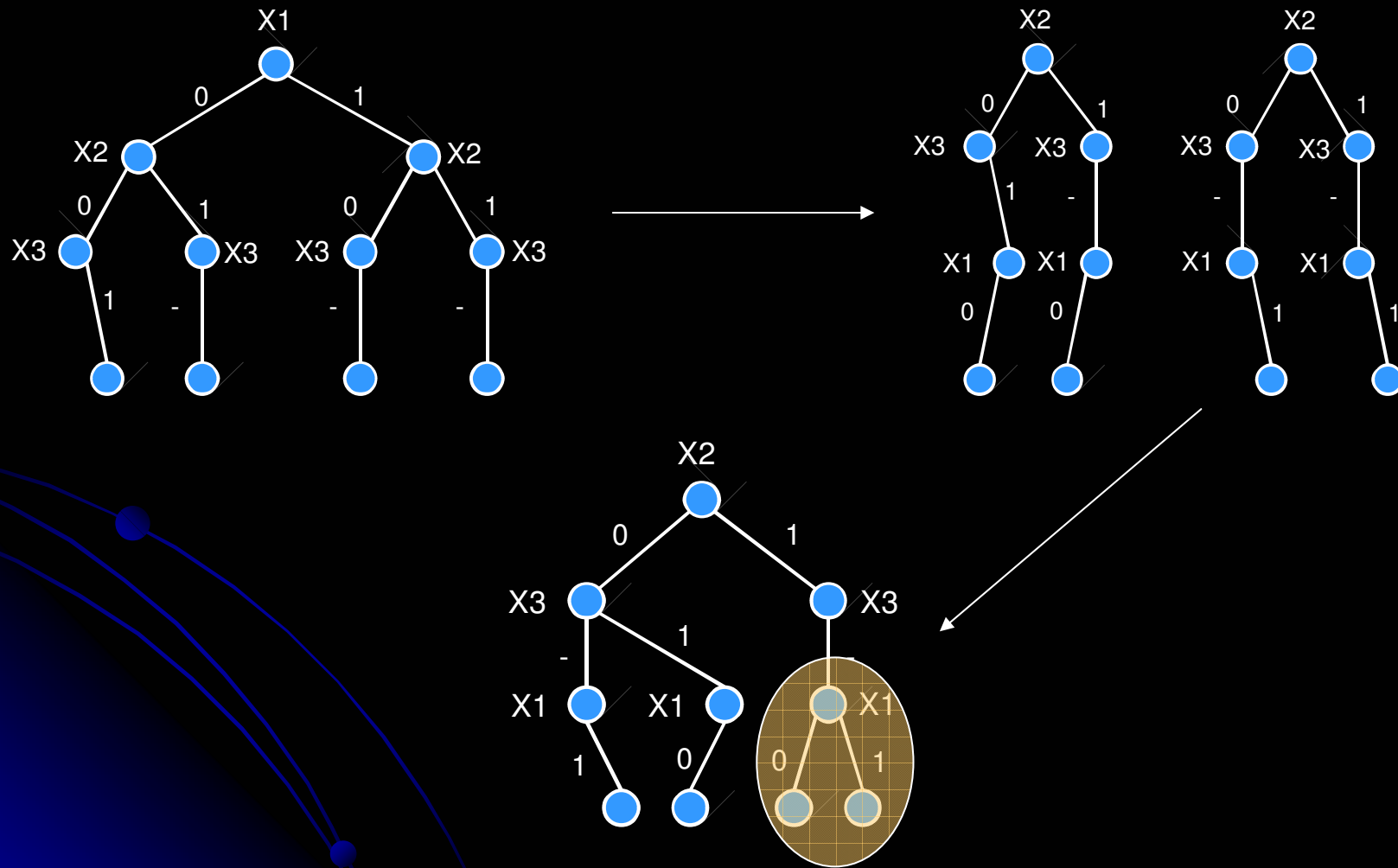


# Tree Rotation

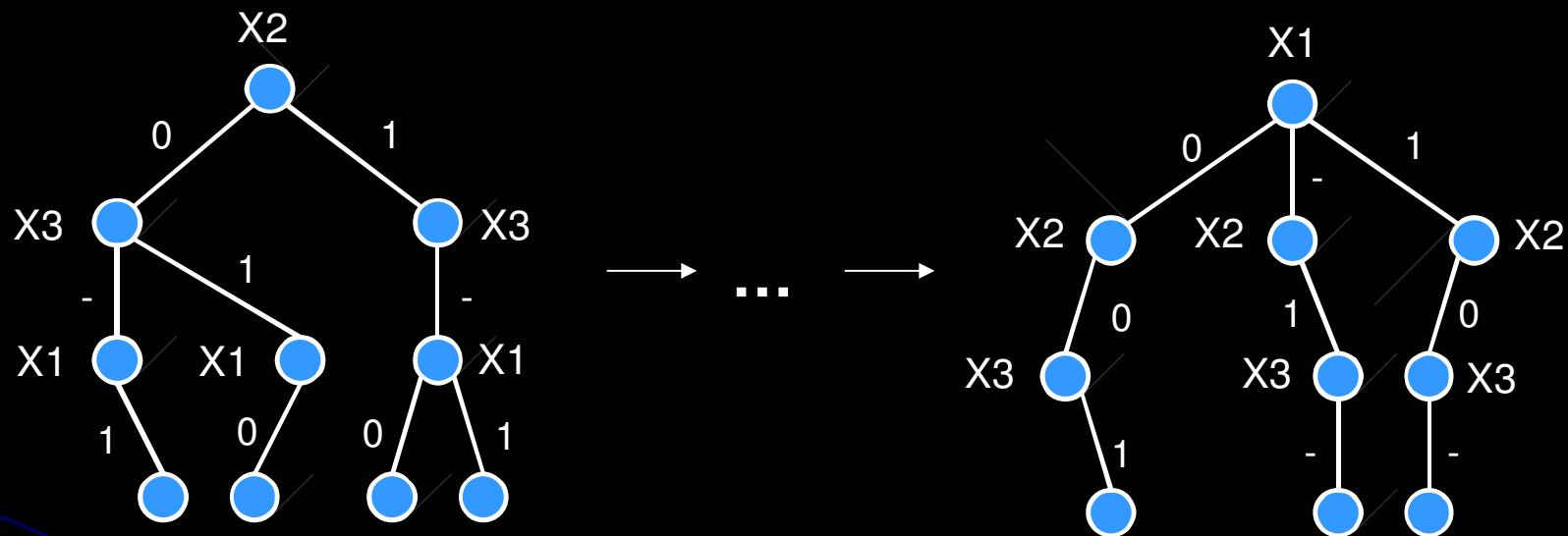
The operations can be performed on leaves only  $\Rightarrow$  the tree must be rotated

1. Cut off the root node  $\rightarrow$  TT is split into three (at most)
2. Append the root variable to all the leaves
3. Merge the trees

# Tree Rotation Example



# TT Minimization Example contd.



10-  
00-  
01-  
11-

001  
-1-  
10-

# Some Experimental Results

Not too positive ... (???)

CNF to DNF conversion results:

Benchmark	Terms			
	Book	MVSIS	ITool	MVSIS
cordic	318.4	3.19	173	1191
cps	5.5			1870
duke2	2.1			452
ex4	2.1			334
ex1010	64			1415
misex3c	11.1		502	508
pdcc	204.8	30.26		897
rd84	1.661	0.53		239
spla	197.931		500	855



# Conclusions

- BoolTool is a tool for doing Boolean operations upon Boolean functions, or, better, Boolean networks
- No such a general tool is known to exist
- It cannot compete in performance with dedicated tools (synthesis tools, SAT solvers)
- Ternary tree minimization algorithm has been developed as a byproduct
- The tool is available to publics at <http://service.felk.cvut.cz/vlsi/prj/BoolTool/>