Multi-level Logic Benchmarks: An Exactness Study

Luca Amaru*,
Mathias Soeken†, Winston Haaswijk†, Eleonora Testa†, Patrick Vuillod*,
Jiong Luo*, Pierre-Emmanuel Gaillardon‡, Giovanni De Micheli†

*Synopsys Inc., Design Group, Sunnyvale, California, USA
†Integrated Systems Laboratory, EPFL, Lausanne, Switzerland
‡LNIS, University of Utah, Salt Lake City, Utah, USA

Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions

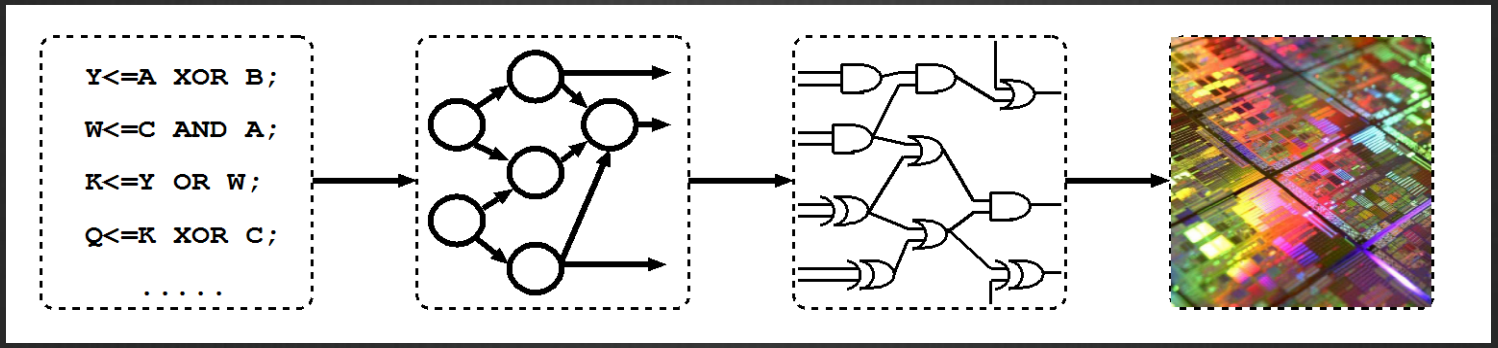
Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions

Benchmarks in EDA

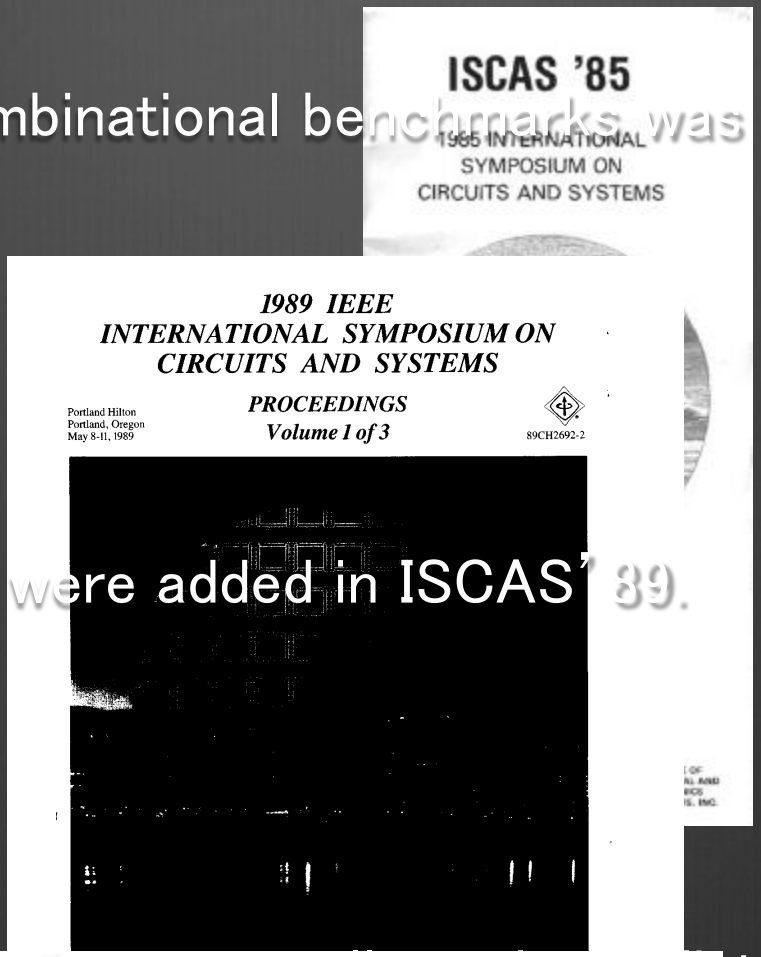
- The EDA community heavily relies on benchmarks to evaluate the performance of academic and commercial tools.

- Logic optimization and synthesis are core EDA tasks where benchmarking is essential.



Historical Background

- The first set of combinational benchmarks was reported in ISCAS' 85.



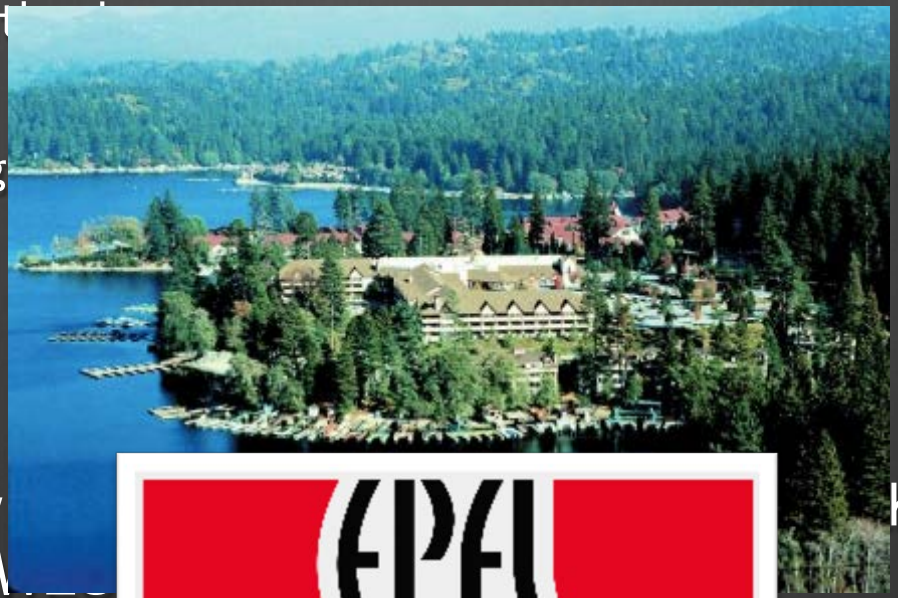
- Sequential circuits were added in ISCAS' 89.

- In 1991, all benc stributed under the maintenance

MCNC®
Connecting North Carolina's Future Today

Historical Background

- After MCNC, many other benchmark suites were proposed for some classes of EDA tasks:
  - High level synt
  - FPGA
  - Physical desig
  - Testing
  - ….

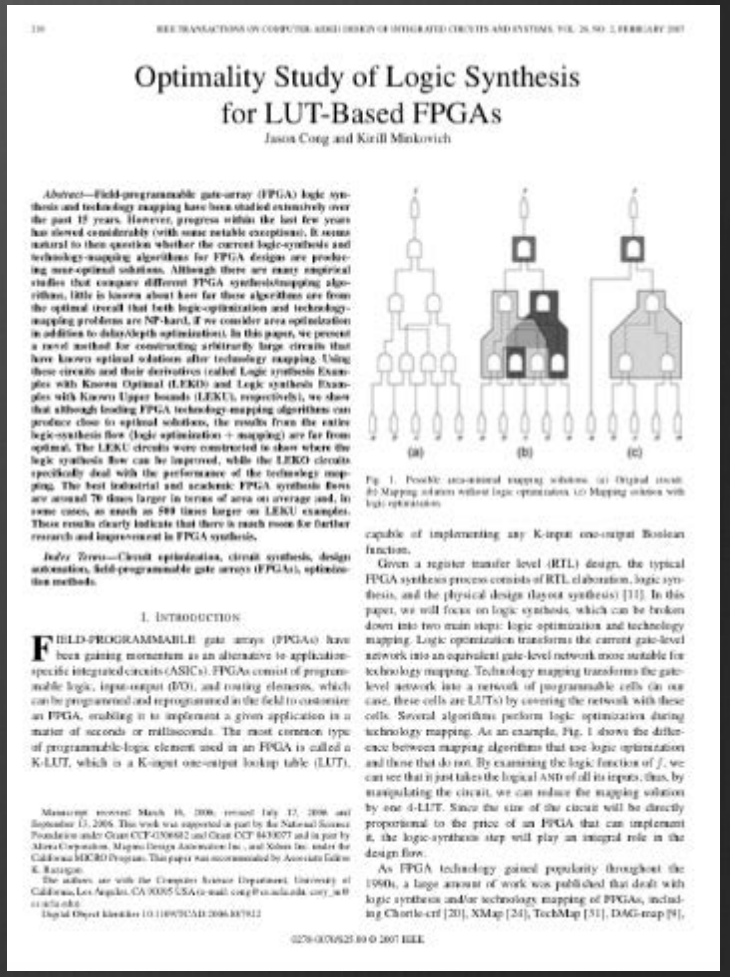


- In 2005, a new hesis was presented at I
- In 2015, a new set benchmarks was presented at IWLS.
- Missing feature: Can we compare against exact synthesis results for multi-level benchmarks?

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Exact-size Benchmarks

- Exact size benchmarks
- TCAD paper 2007
- LEKO and LEKU
  - LEKO: known optimum
  - LEKU known upper-bound
- Exactness idea: replicate a small circuit, in a special way, with a known optimum solution
  - The composite circuit is also optimal
- LEKU = LEKO collapsing + gate decomposition



Optimality Study of Logic Synthesis for LUT-Based FPGAs

Jason Cong and Kirill Minkovich

Abstract—Field-programmable gate-array (FPGA) logic synthesis and technology mapping have been studied extensively over the past 15 years. However, progress within the last few years has slowed considerably (with some notable exceptions). It seems natural to then question whether the current logic-synthesis and technology-mapping algorithms for FPGA designs are producing near-optimal solutions. Although there are many empirical studies that compare different FPGA synthesis/mapping algorithms, little is known about how far these algorithms are from the optimal (recall that both logic-optimization and technology-mapping problems are NP-hard). If we consider area optimization in addition to delay/depth optimization). In this paper, we present a novel method for constructing arbitrarily large circuits that have known optimal solution after technology mapping. Using these circuits and their derivatives (called Logic synthesis Examples with Known Optimal (LEKO) and Logic synthesis Examples with Known Upper bounds (LEKU), respectively), we show that although leading FPGA technology-mapping algorithms can produce close to optimal solutions, the results from the entire logic-synthesis flow (logic optimization + mapping) are far from optimal. The LEKU circuits were constructed to show where the logic synthesis flow can be improved, while the LEKO circuits specifically deal with the performance of the technology mapping. The best industrial and academic FPGA synthesis flows are around 70 times larger in terms of area on average and, in some cases, as much as 500 times larger on LEKU examples. These results clearly indicate that there is much room for further research and improvement in FPGA synthesis.

Index Terms—Circuit optimization, circuit synthesis, design automation, field-programmable gate arrays (FPGAs), optimization methods.

I. Introduction

FIELD-PROGRAMMABLE gate arrays (FPGAs) have been gaining momentum as an alternative to application-specific integrated circuits (ASICs). FPGAs consist of programmable logic, input-output (I/O), and routing elements, which can be programmed and reprogrammed in the field to customize an FPGA, enabling it to implement a given application in a matter of seconds or milliseconds. The most common type of programmable-logic element used in an FPGA is called a K-LUT, which is a K-input one-output lookup table (LUT),

Manuscript received March 10, 2006; revised July 12, 2006 and September 13, 2006. This work was supported in part by the National Science Foundation under Grant CCF-0306682 and Grant CCF-0430077 and in part by Altera Corporation, Magma Design Automation Inc., and Xilinx Inc. under the California MICRO Program. This paper was recommended by Associate Editor K. Rajagopal.

The authors are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; cory_m@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2006.887922

Fig. 1. Possible area-minimal mapping solutions. (a) Original circuit. (b) Mapping solution without logic optimization. (c) Mapping solution with logic optimization.

capable of implementing any K-input one-output Boolean function.

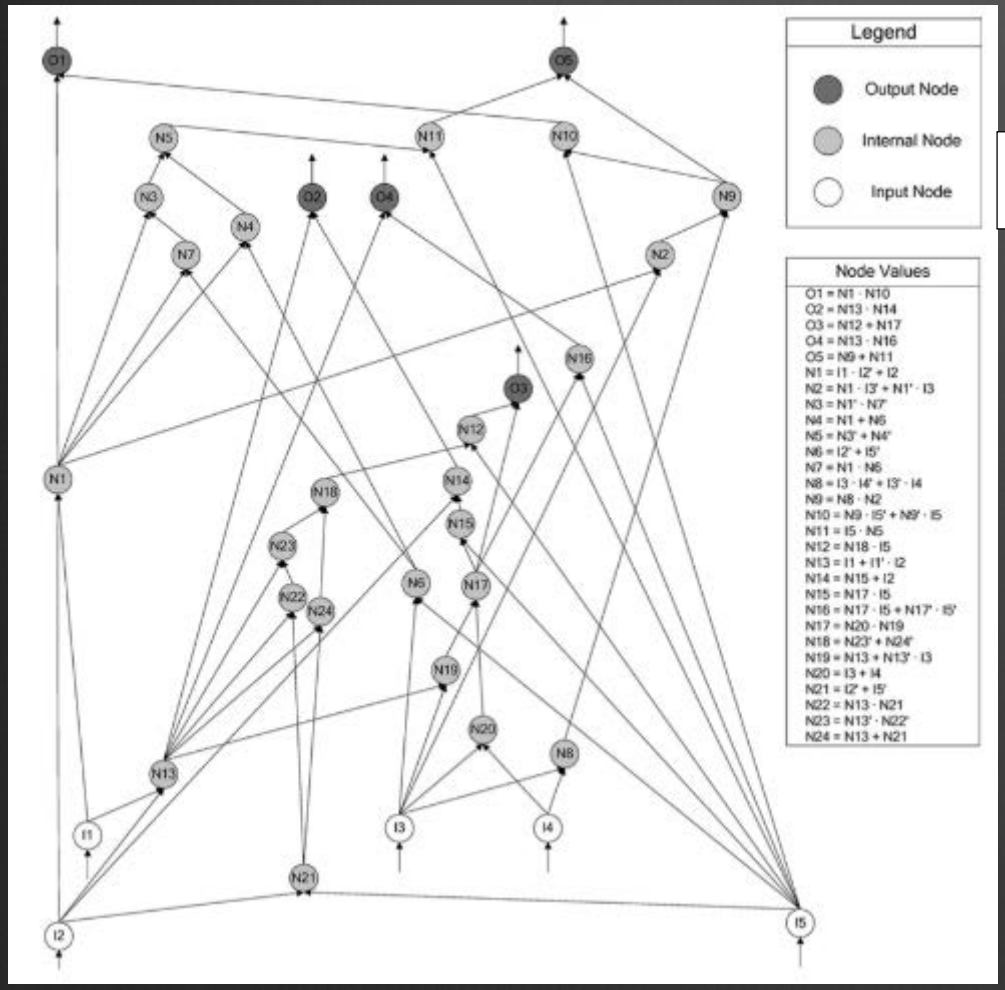
Given a register transfer level (RTL) design, the typical FPGA synthesis process consists of RTL elaboration, logic synthesis, and the physical design (layout synthesis) [11]. In this paper, we will focus on logic synthesis, which can be broken down into two main steps: logic optimization and technology mapping. Logic optimization transforms the current gate-level network into an equivalent gate-level network more suitable for technology mapping. Technology mapping transforms the gate-level network into a network of programmable cells (in our case, these cells are LUTs) by covering the network with these cells. Several algorithms perform logic optimization during technology mapping. As an example, Fig. 1 shows the difference between mapping algorithms that use logic optimization and those that do not. By examining the logic function of f, we can see that it just takes the logical AND of all its inputs; thus, by manipulating the circuit, we can reduce the mapping solution by one 4-LUT. Since the size of the circuit will be directly proportional to the price of an FPGA that can implement it, the logic-synthesis step will play an integral role in the design flow.

As FPGA technology gained popularity throughout the 1980s, a large amount of work was published that dealt with logic synthesis and/or technology mapping of FPGAs, including Chortle-crf [20], XMap [24], TechMap [31], DAG-map [9],

Exact-size Benchmarks

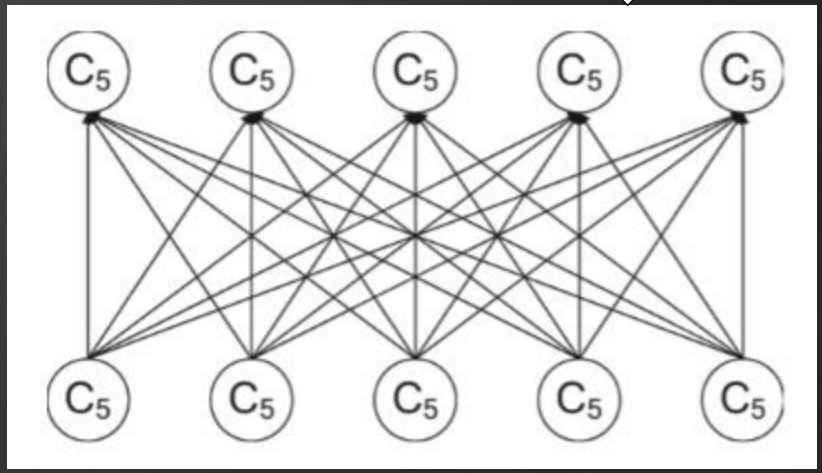
- LEKO Example:

C_5 circuit



Composite circuit:

LEKO G_{25}



Motivation For This Work

- If we want to measure the ultimate performance of logic synthesis heuristics, we need exact results for multi-level logic benchmarks
- Exact results and benchmarks for size do exist: LEKO & LEKU
- In this work, we show how to build exact depth benchmarks
  - Measure the ultimate performance of delay-oriented synthesis algorithms and tools
- We provide a non-replicative method to build exact depth, multi-level circuits, with:
  - Non-trivial functionality
  - Non-monotone
  - No disjoint support decomposition

Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions

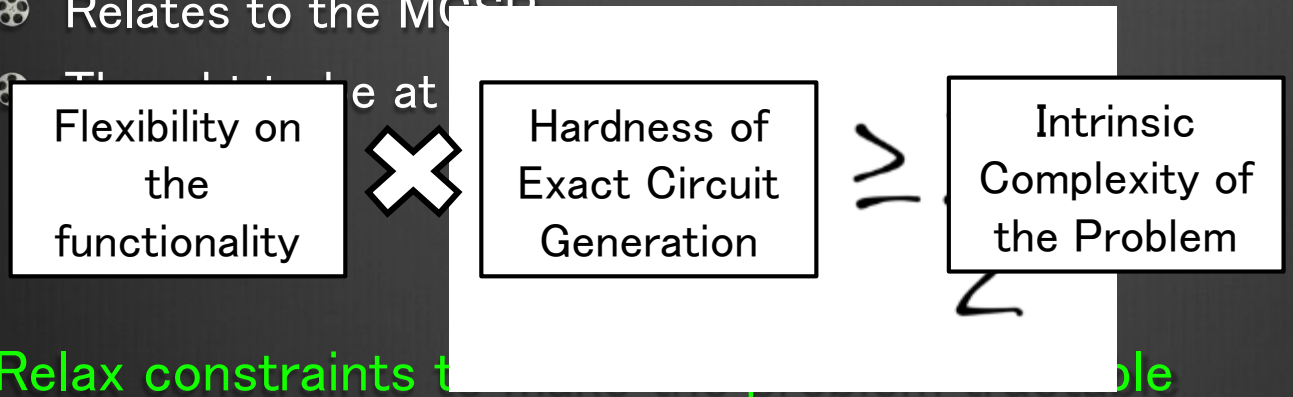
Exact Benchmarks

- Problem: we want to measure the efficiency of heuristic delay optimization techniques, for large circuits with known optimum results
  - Thousands of I/O
  - Tens of thousands of gates
  - Potentially small delay (but large delay as starting point)

- Solution: build synthetic benchmarks that are provably optimal
  - Avoid trivial cases
  - Avoid monotone circuits
  - Avoid disjoint support decomposable circuits
  - Add extra complexity

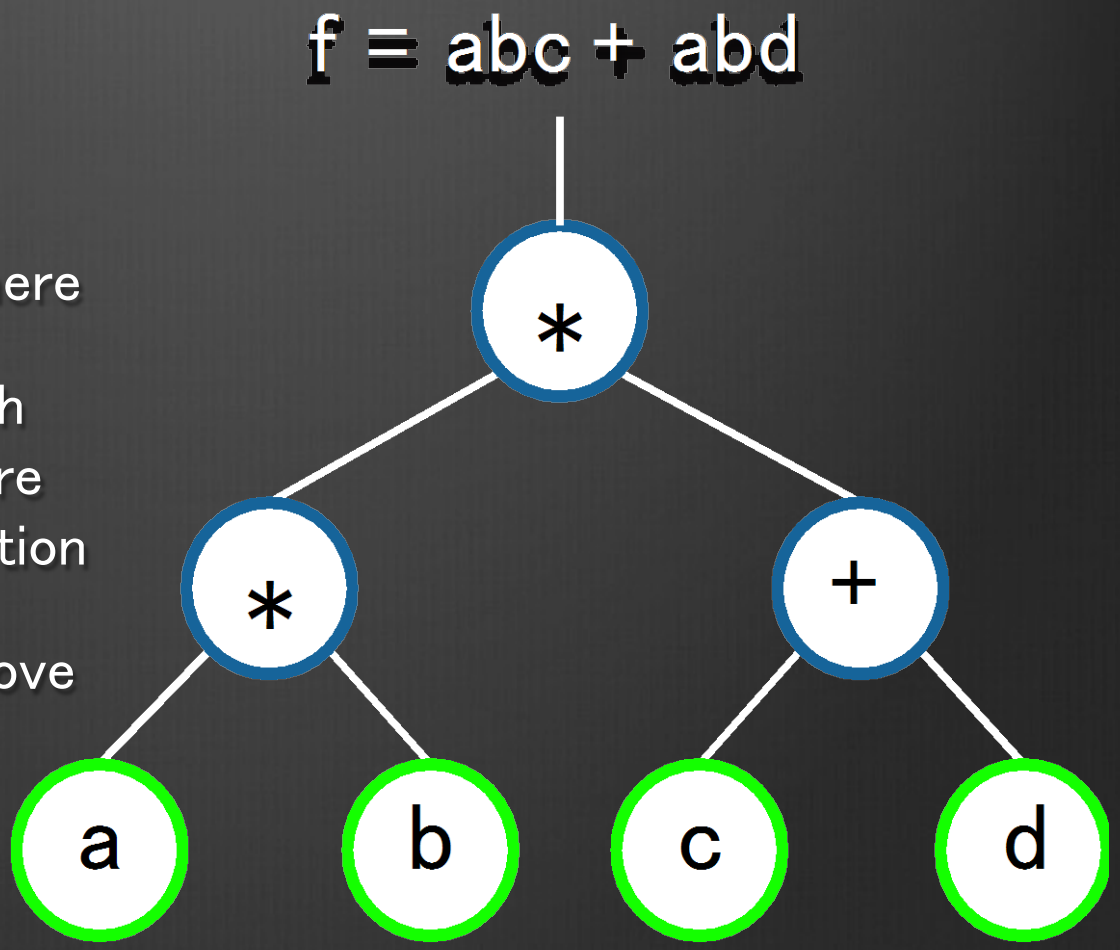
Why Synthetic Exact Benchmarks?

- Question: why synthetic exact benchmarks?
- Generating optimum circuits for arbitrary functions is an intractable problem
  - Relates to the MCSP
  - Theoretically intractable at
- Relax constraints t ble
  - Impose complexity constraints on the function but not on the actual functionality
  - The logic synthesis version of Heisenberg's uncertainty principle:
    - The more we want to know the functionality a priori, the harder it becomes to know the exact circuit



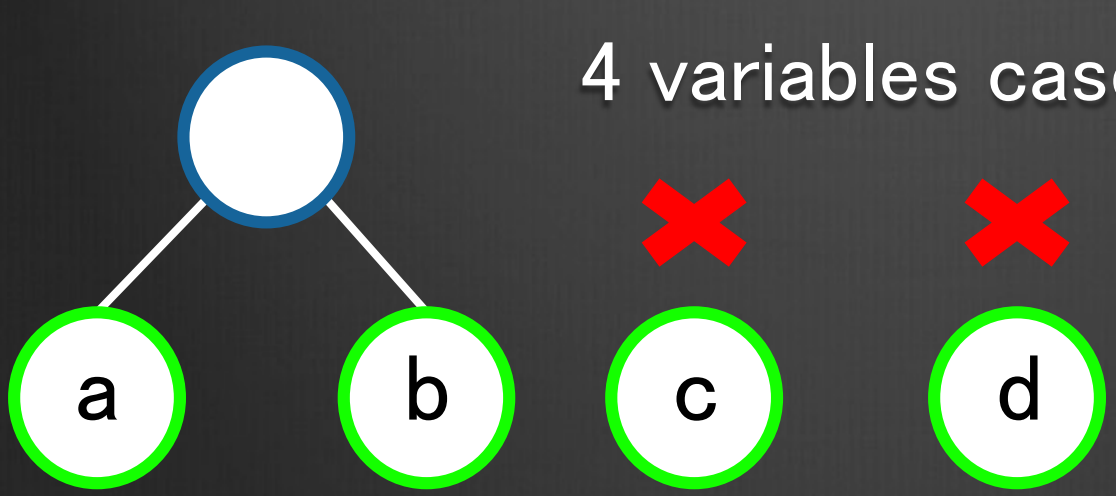
Balanced-tree Exact Circuits

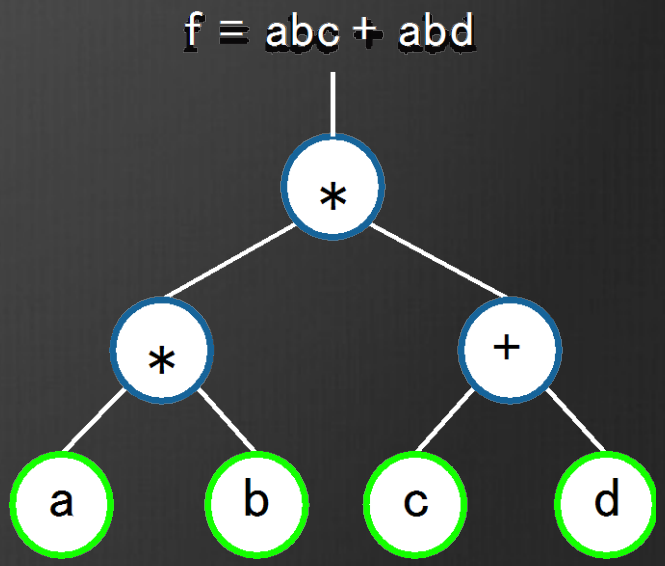
- How to build exact-depth multi-level circuits?
- Let's start from a simple concept and move from there
- Balanced-tree circuits with distinct inputs as leaves are depth optimal by construction
- Intuitively true, easy to prove



Are Balanced-tree Circuits Depth Optimal?

- Proof by contradiction:
- A balanced tree is a *n* level implementation for a 2^n variables function.
- Let's assume it is possible to implement the same function in *n−1* levels

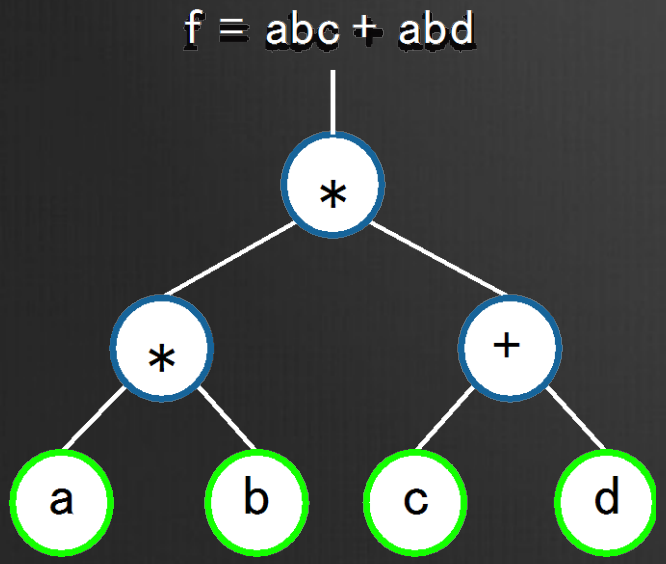




- The function realized cannot depend on all 2^n variables: hence the contradiction.

Construction Algorithm: v1.0

- Simple algorithm:
- Generate a binary tree with n levels, 2^n leaves, $2^n - 1$ nodes
- Populate the nodes (randomly) with AND/OR binary operators
- Populate the leaves with distinct primary inputs
- The root of the tree is the primary output



- Problem: the functions implemented are trivial
- Only monotone circuits
- Specialized synthesis algorithms can easily identify such structures

Construction Algorithm: v1.5

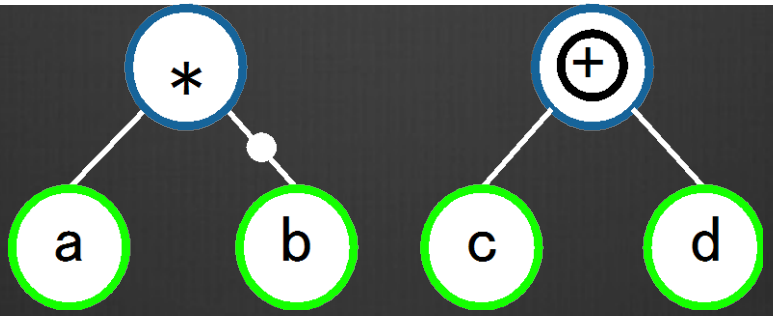
- Breaking the monotone property:
- Int
- NA
- Co
- En
  op

te

Algorithm 1 Generation of depth-optimal multi-level circuits with disjoint support.

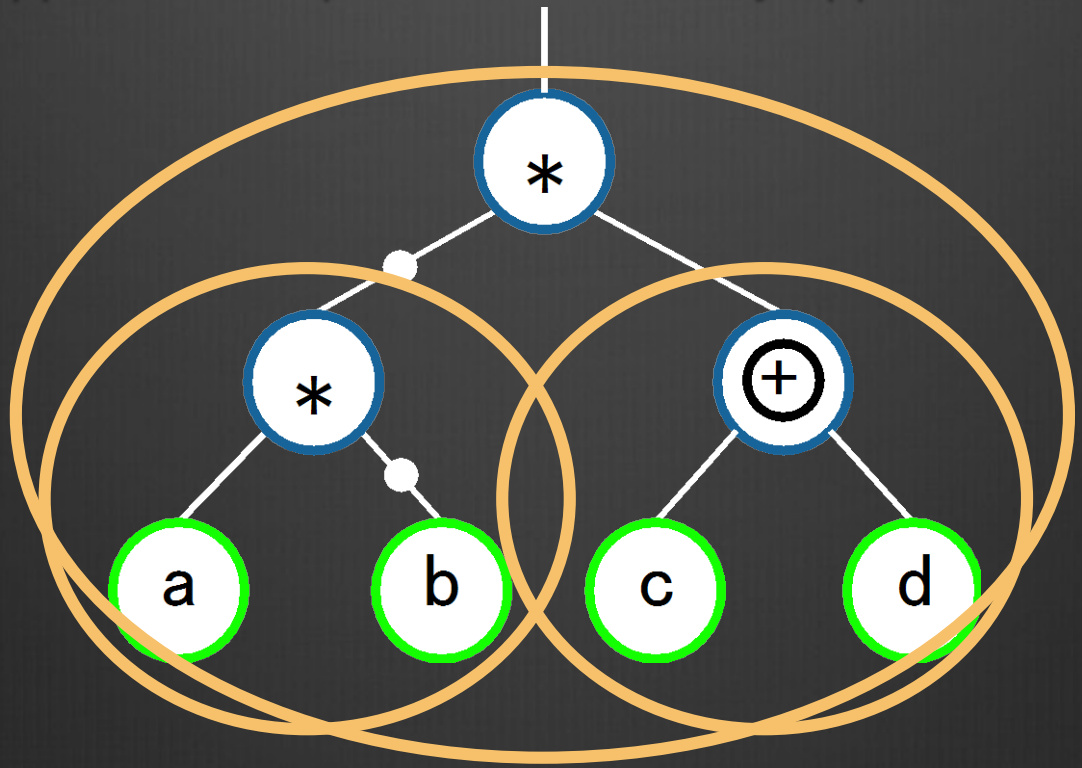
INPUT: Complexity measure n
OUTPUT: Depth-optimal circuit with 2^n inputs.

```
create empty balanced binary tree with n levels;
for each node n do
    assign n a random binary operator;
end for
enforce the presence of at least one binate operator;
```



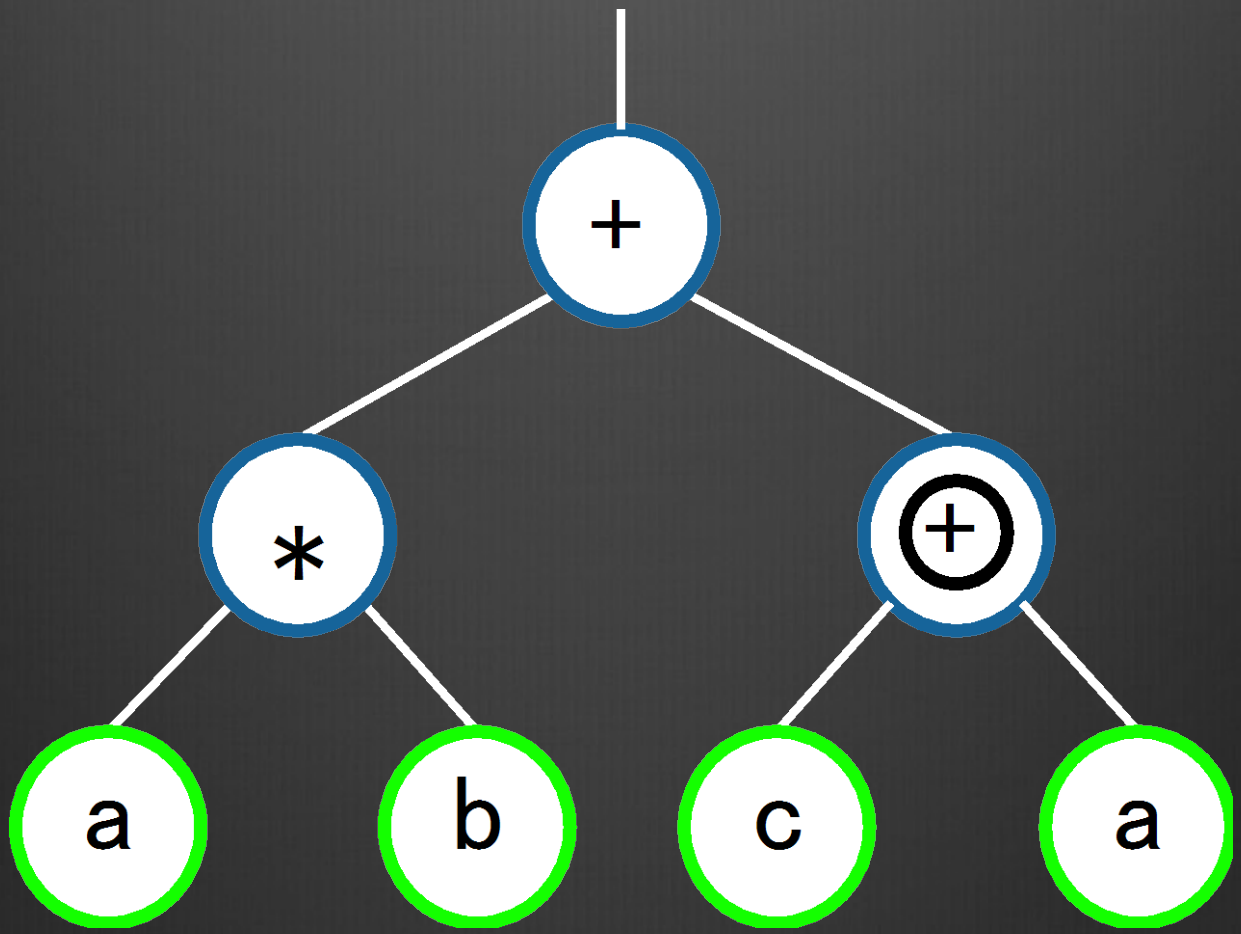
Disjoint Support Decomposition

- Addressed problem: the functions implemented are not monotone
- Remaining problem: the functions implemented are still easy to synthesize
- Disjoint support decomposition is natively applicable here



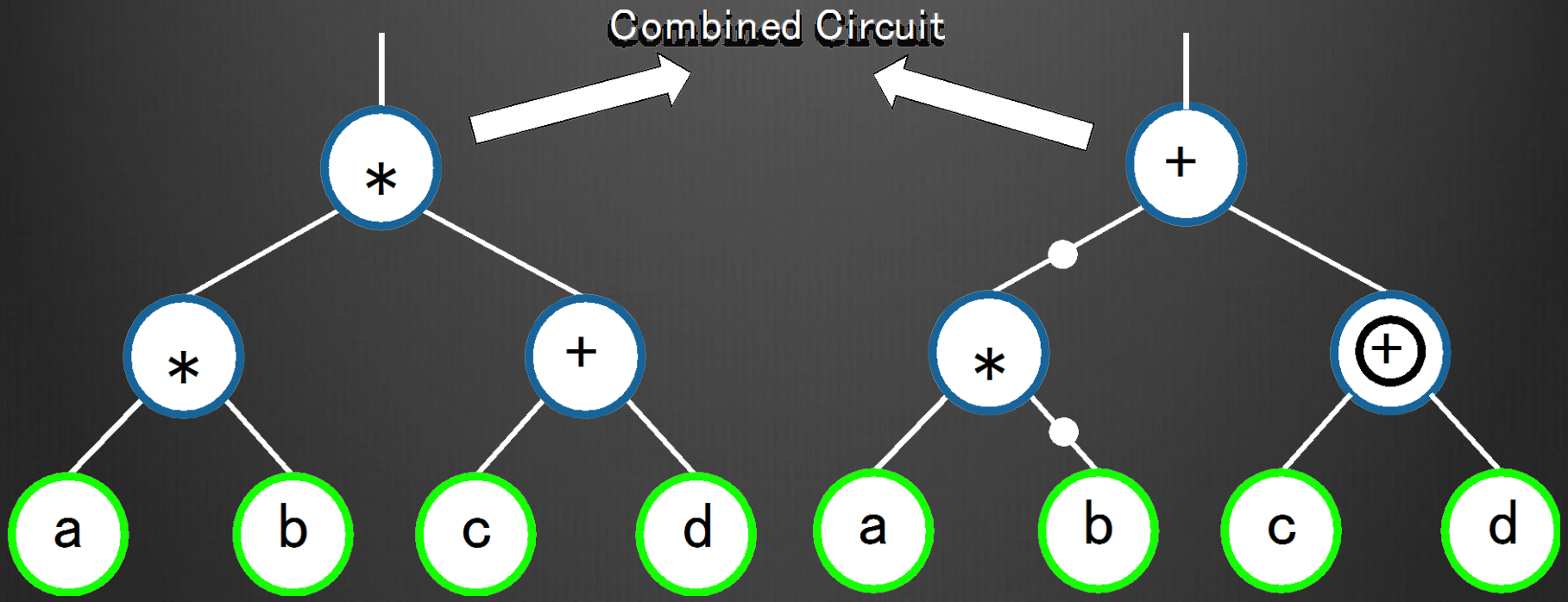
Breaking the DSD Property

- We need portions of logic with possibly joint support



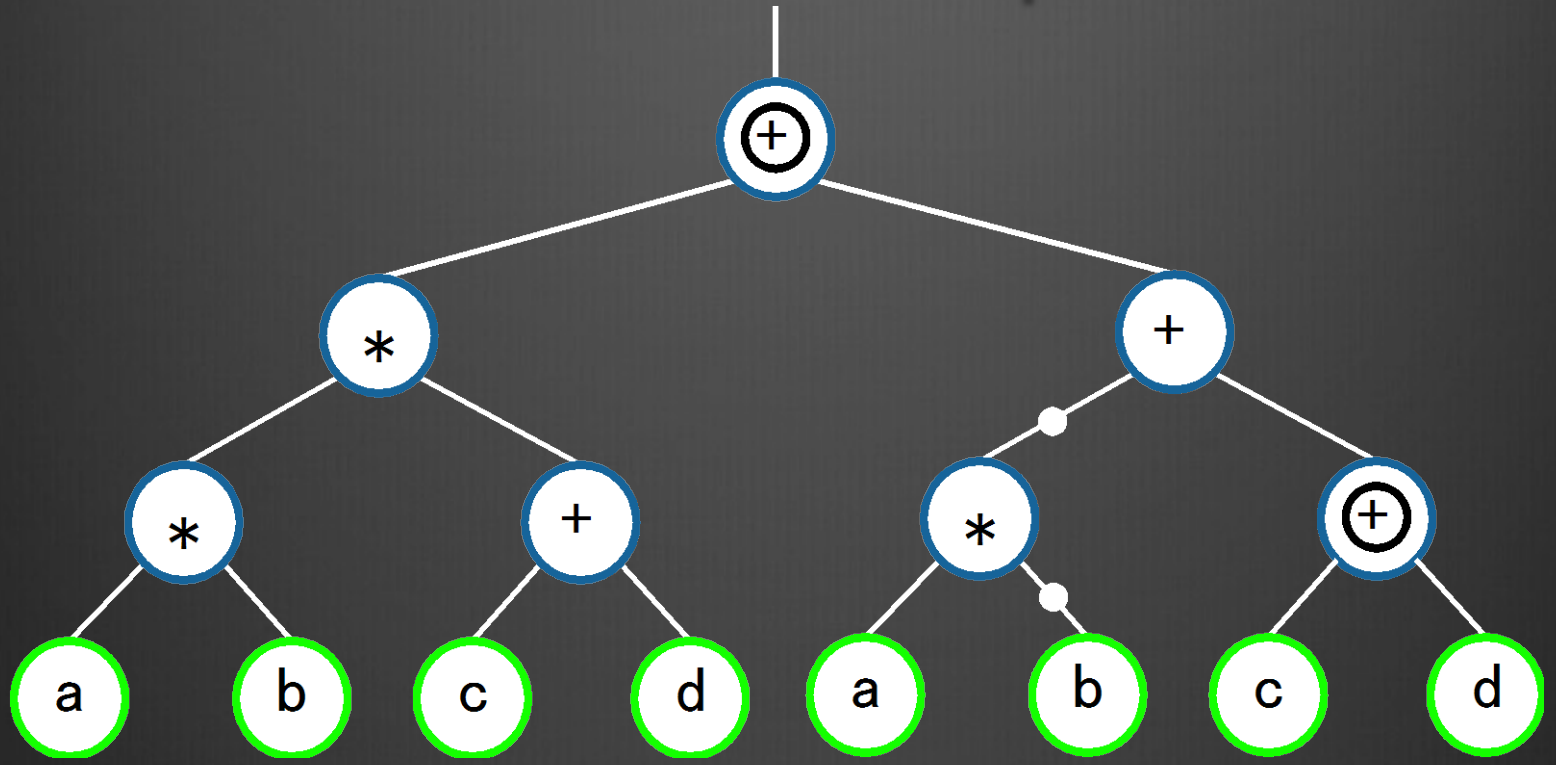
Breaking the DSD Property

- Idea: merge depth optimal tree circuits with shared support!



- Each tree is generated by the previous algorithm
  - Functions are non-monotone
- The two trees can be combined with a top binary operator
  - XOR/XNOR are preferable

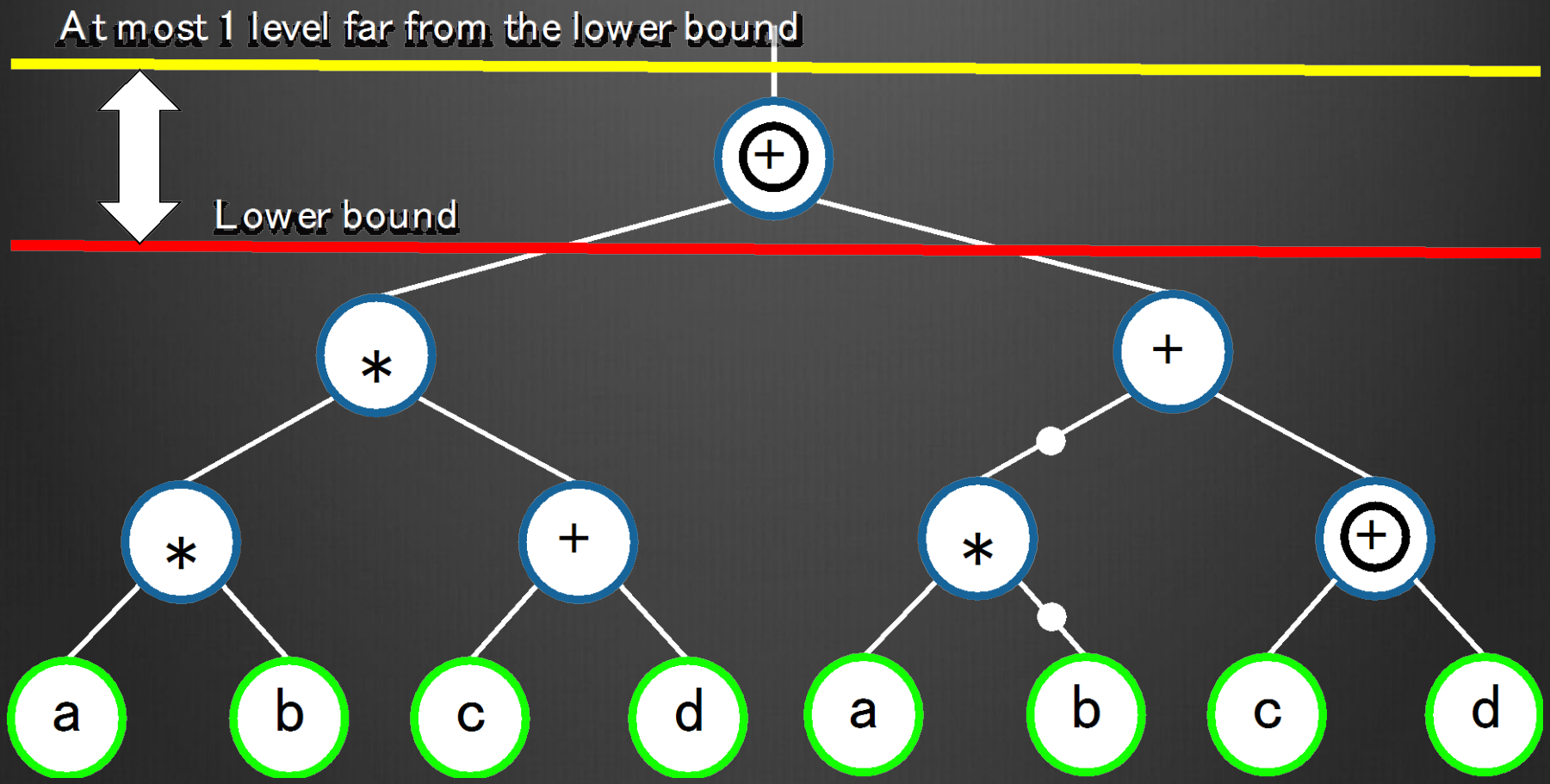
Benchmark Properties



- The circuit obtained has:
- 2^n inputs
- $n+1$ levels
- $2^n - 1$ nodes

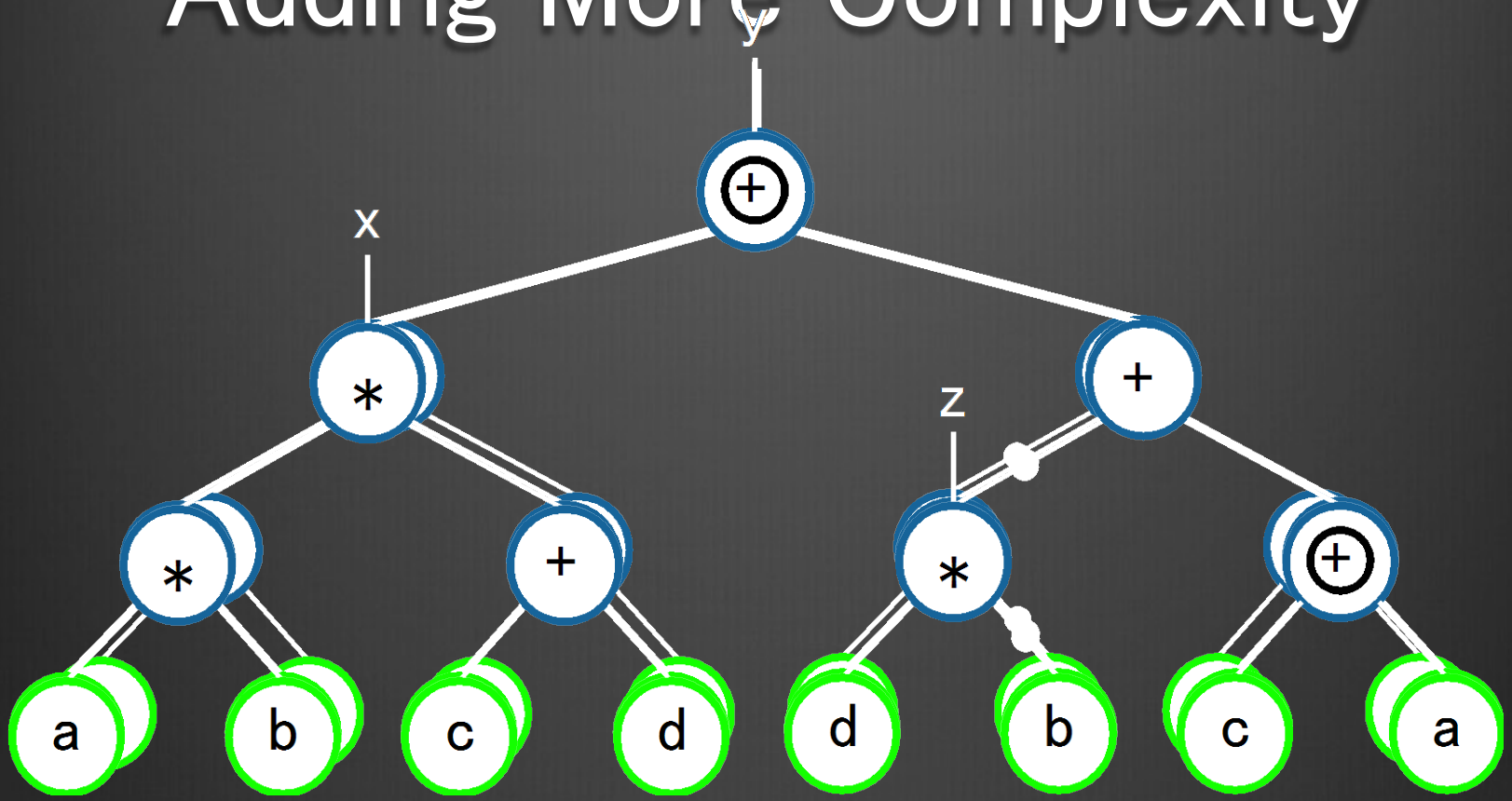
- The function implemented is:
- Non-monotone
- Not directly DSD decomposable

Exactness



- This is true if the cardinality of the functional support is 2^n
- More about this in a moment

Adding More Complexity

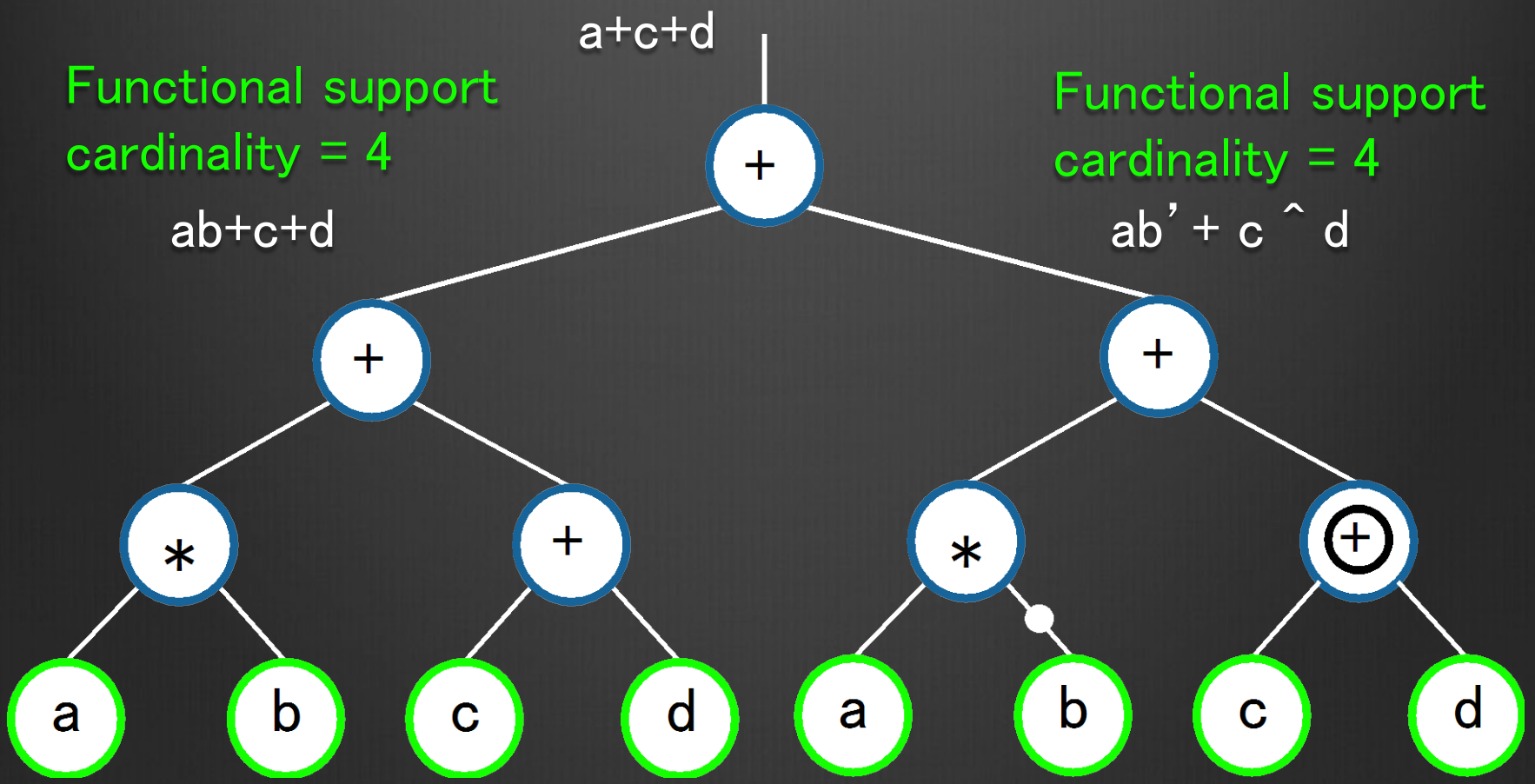


- Can we make the circuit more complex?
- Swap inputs of one tree
- Add primary outputs as internal nodes

BAD NEWS: we may decrease the cardinality of the functional support

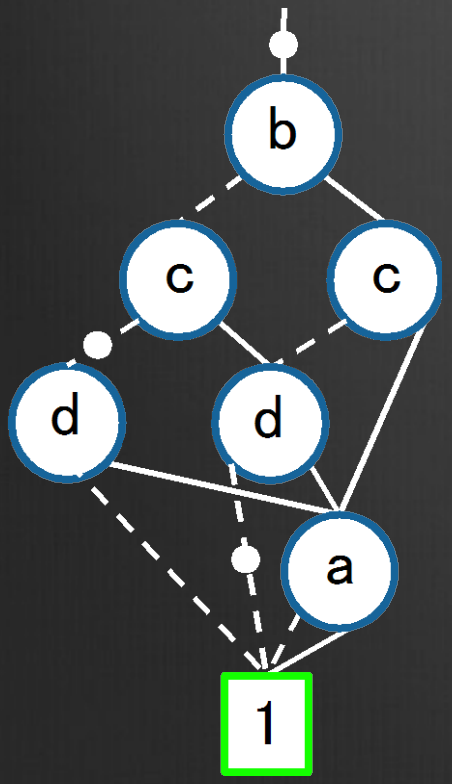
Functional Support

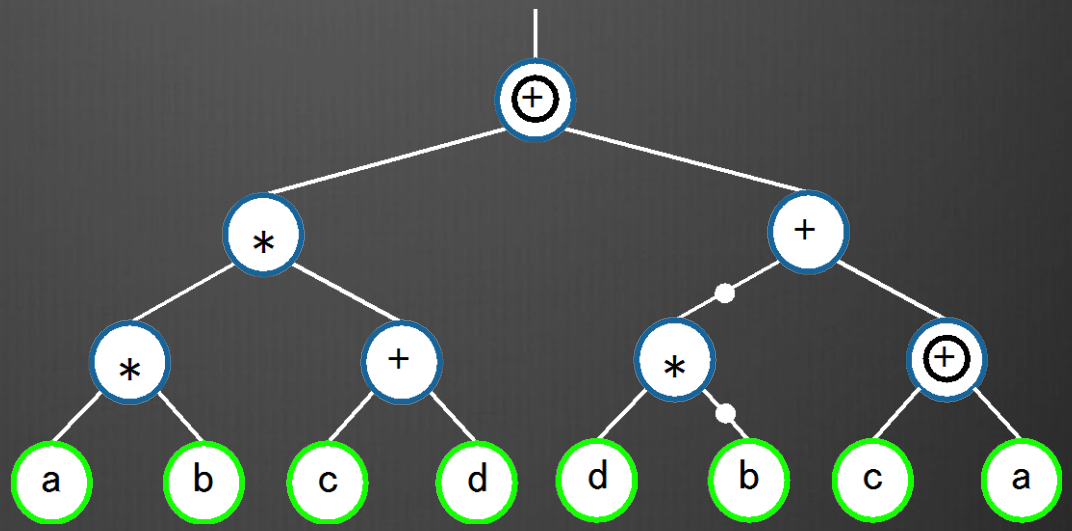
- Simple example:



Verifying the Functional Support

- Solution: Check the functional support of the composite circuit
  - BDD techniques
  - SAT techniques






```
abc 01> read example.blif; strash; print_supp -s
Total func supps   =         4.
Total struct supps =         4.
Sat runs SAT       =         0.
Sat runs UNSAT     =         0.
Simulation    =      0.00 sec
Traversal     =      0.00 sec
Fraiging      =      0.00 sec
SAT           =      0.00 sec
TOTAL         =      0.00 sec
abc 03>
```

Construction Algorithm: v2.0

- Final construction algorithm:

Algorithm 2 Generation of depth-optimal multi-level circuits with joint support.

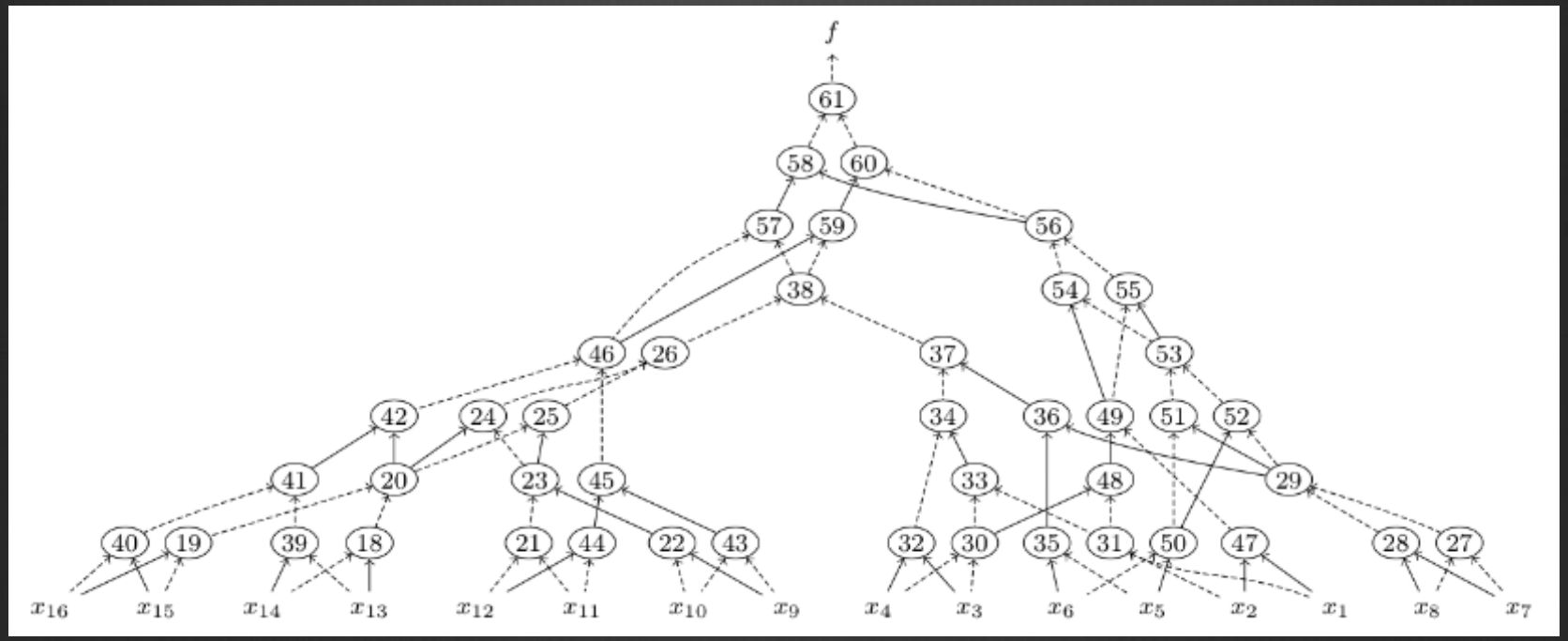
INPUT: Complexity measure n
OUTPUT: Depth-optimal circuit with 2^n inputs.

A = Algorithm 1(n);
B = Algorithm 1(n);
share primary inputs of A and B;
create node n that joins roots of A and B;
assign node n to a random binary operator;
set n as primary output of the composite circuit;
verify the functional support of the composite circuit;

- Hard to synthesize circuits (non-monotone, non-DSD)
- At most 1 level far from the optimum

Exact Benchmark Example

- Example circuit with complexity measure = 4
  - 16 inputs
- Result strashed into AIG
- The depth optimality is not guaranteed after strashing



Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions

Experiments: Initial Circuits

- Starting implementation: BDD-collapsing of exact circuits

2^n inputs

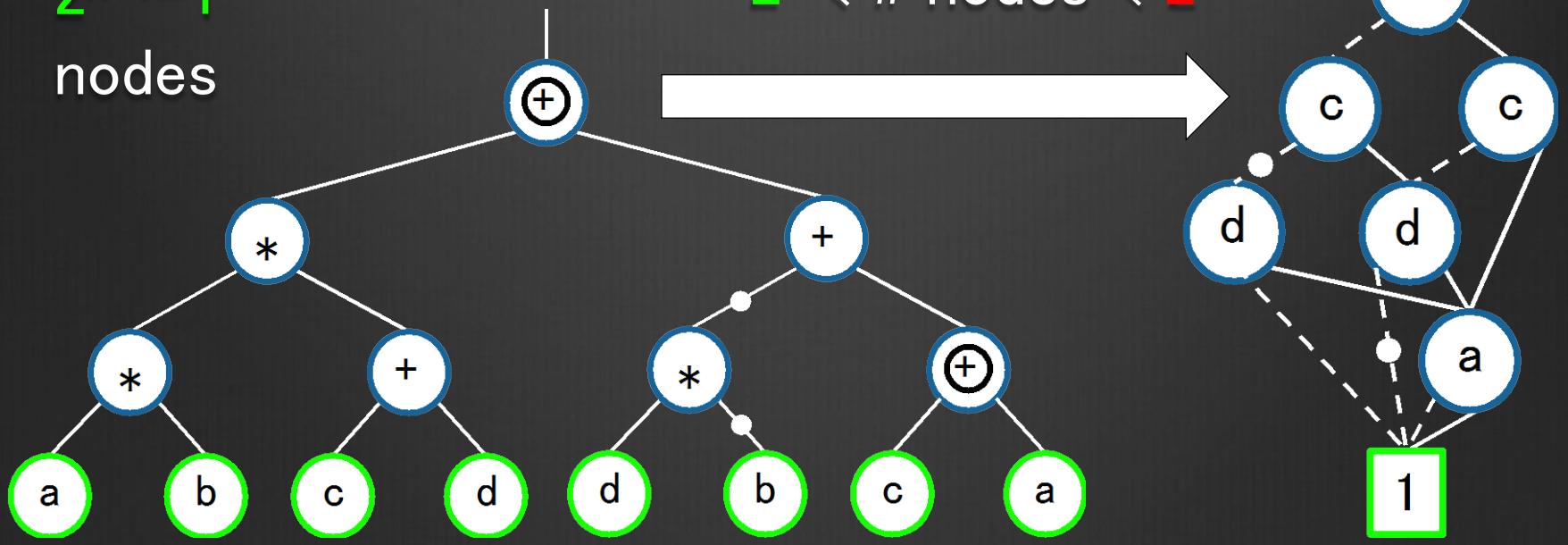
$n+1$ levels

$2^{n+1}-1$ nodes

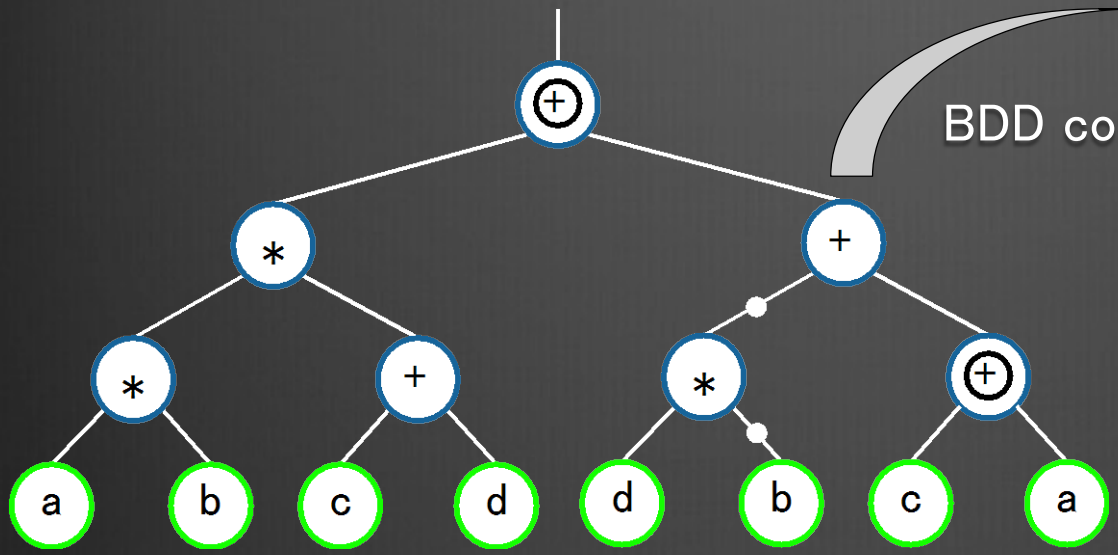
2^n inputs

2^n levels

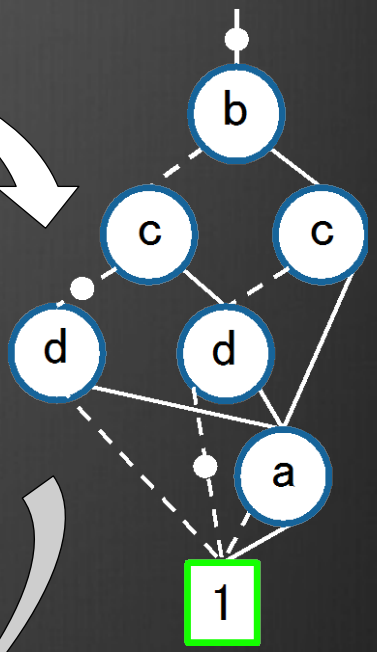
$2^n < \#\text{ nodes} < 2^{2^{(n-1)}}$



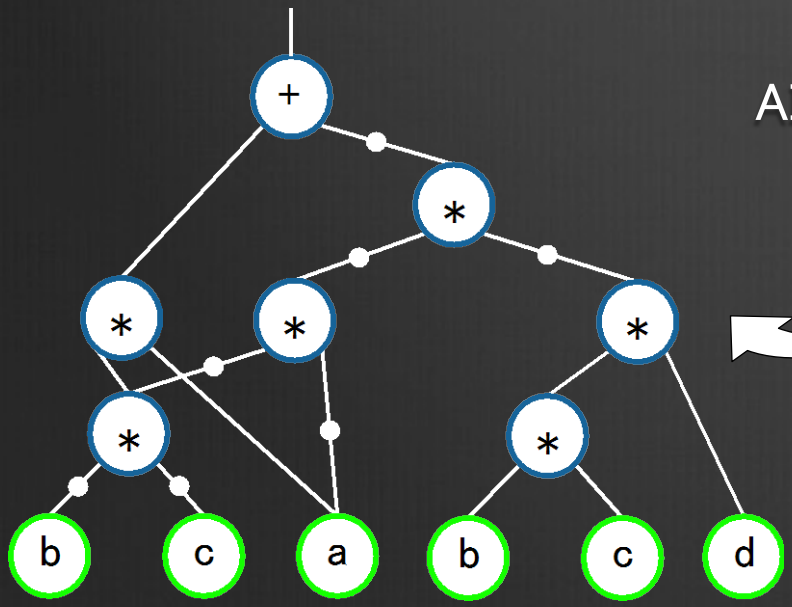
Experiments: Sample Testing Flow



BDD collapsing



AIG optimization



Mapping onto:

AND (unit cost) OR (unit cost)

XOR (unit cost) INV (free)

Experimental Results

- Use of different delay synthesis techniques
  - Starting point: collapsed BDD
- DSD, AIG, MIG, AIG–strashing of the exact circuit

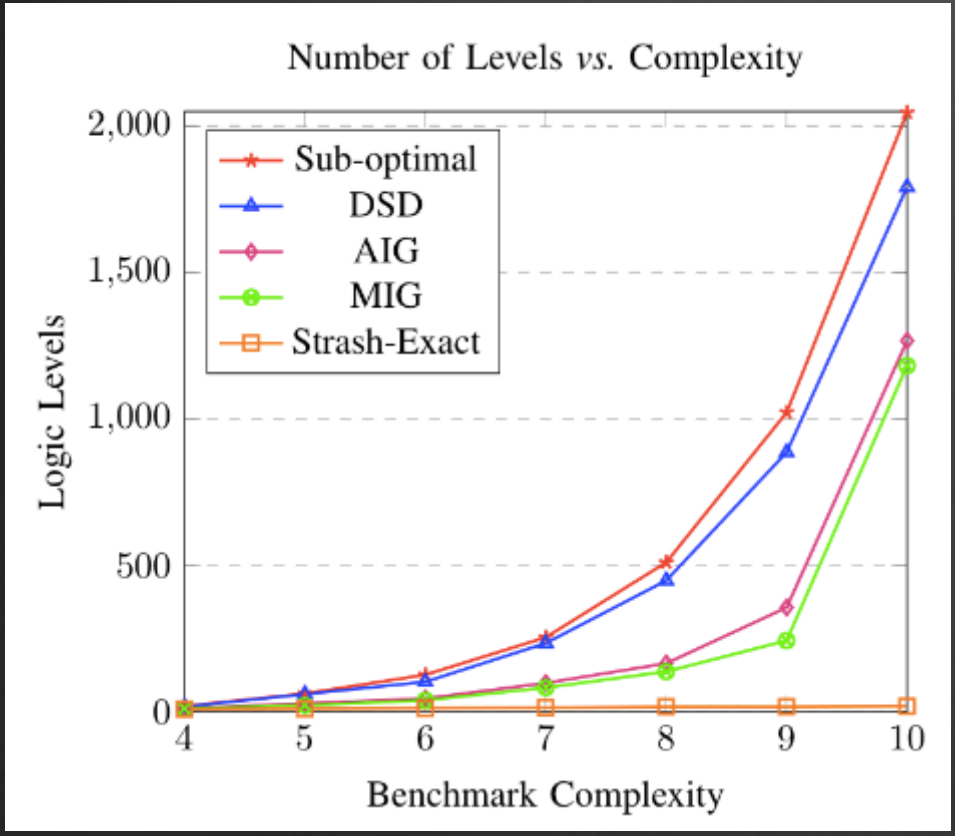


TABLE I
SYNTHESIS EXPERIMENTS: LOGIC DEPTH

Complexity	Sub-opt.	DSD	AIG	MIG	St-Exact
4	19	17	11	11	9
5	62	60	28	22	10
6	126	102	45	39	12
7	254	234	98	82	14
8	510	448	165	137	17
9	1022	886	356	243	17
10	2046	1792	1267	1182	19

- Exponential gap between known optimal results and heuristics

Experimental Results

- Same setup, reporting size values:

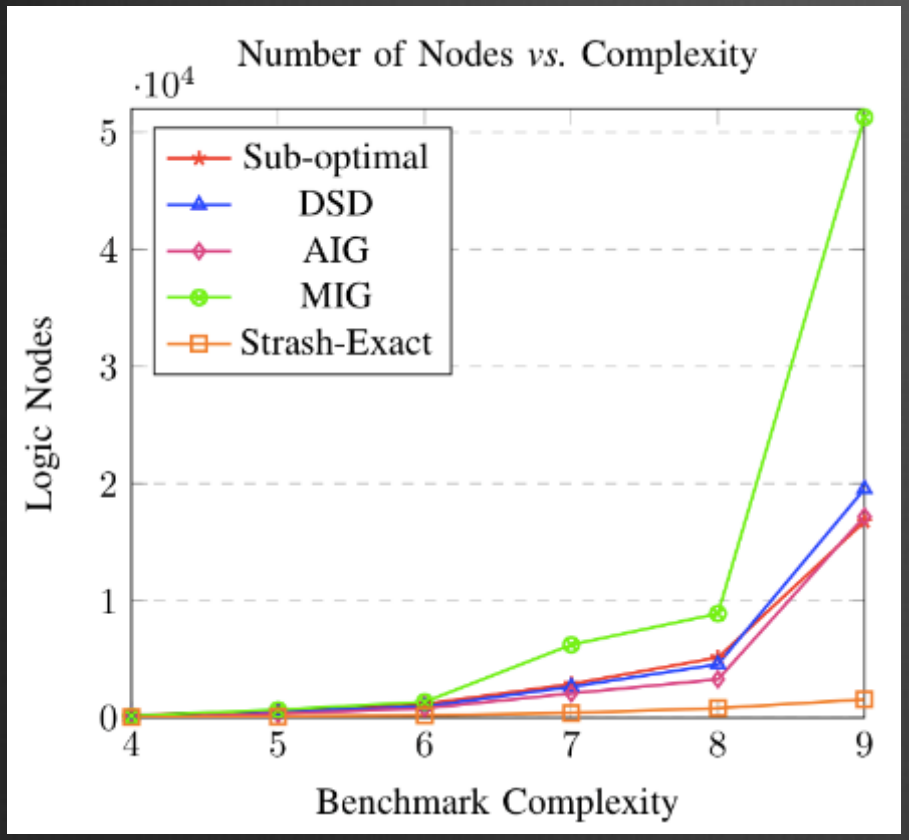


TABLE II
SYNTHESIS EXPERIMENTS: LOGIC SIZE

Complexity	Sub-opt.	DSD	AIG	MIG	St-Exact
4	235	175	154	164	**51**
5	524	517	331	686	**97**
6	1176	961	801	1351	**204**
7	2881	2648	2081	6236	**411**
8	5145	4563	3218	8893	**818**
9	16752	19541	17184	51278	**1564**
10	634806	468735	269432	275052	**3110**

- Exponential gap also for size
- Limit for scalability: BDD collapsing
  - 1024 inputs already has a 0.6M nodes BDD
  - After that, collapsing becomes difficult and BDD-size is out of scale

Outline

- Motivation and Background
- How to Build Exact (Multi-level) Benchmarks
- Measuring the Exact-Heuristic Gap in Synthesis
- Conclusions

Conclusions and Future Research

- We presented a method to generate exact-depth multilevel benchmarks
  - Non-trivial functionality: non-monotone, non-DSD
  - Benchmark generation is efficient
    - Circuits with less than 2^{10} inputs can be generated in matter of seconds
    - Circuits with more than 2^{10} inputs can be generated in matter of minutes
    - Bottleneck is functional support check
- We proved an exponential gap between known optimal results and heuristics
  - More research is still needed in logic synthesis!
- Future research will consider:
  - More control on the implemented logic function in the exact benchmark
    - Possibility to embed specific functions, or function classes, frequently appearing in practical designs
  - More scalable collapsing
    - Hybrid BDD/SAT based collapsing

Questions?

Thank you for your attention!