A Prudent Approach to Benchmark Collection

Jan Schmidt, Petr Fišer
Czech Technical University in Prague
email: schmidt@fit.cvut.cz, fiserp@fit.cvut.cz

Abstract

We present experimental evidence that logic synthesis procedure, especially those based on resynthesis, do net perform well when the original (designer-given) structure of input description is lost. As such performance has not been observed otherwise, we must conclude that such operation is outside of the intended range, and that synthesis examples with their original structure lost are not valid for evaluation of synthesis procedures. We also outline other causes that may render an example invalid. We, however, document that such losses did occur with circuit examples circulating in the logic synthesis community. Therefore, we have to suggest what constitutes prudence in examples collection.

1 Introduction

Electronic Design Automation (EDA) tools, or more precisely, their automated synthesis procedures, are complex layered heuristics. The only way to evaluate them is the way of experiments. For such evaluations, collections of instances (benchmark sets) have been published and are in constant use. The oldest such sets still active are from mid-eighties [4], the last one we are aware of is recent [20]. During that time, almost everything in logic synthesis changed. Circuit implementation moved from Programmable Logic Arrays (PLA) to multilevel gate structures and Look-Up Tables (LUTs). Logic synthesis procedures based on Disjunctive Normal Form, its minimization, and decomposition gave way to iterative resynthesis. Yet, even the oldest benchmark sets are still in use and cited in research papers.

Today, a five-percent improvement in experimental evaluation on such examples is considered worth a conference contribution [32]. On the other end of spectrum, rather alarming results have been obtained experimentally [7][12]. The examples used were not benchmarks, but circuits similar to those met in practice.

The question is, whether these results – good or bad – are significant for the 'primary customer', which is practical design. At the first glance, the answer is yes, the results are significant because the examples are practical designs. Under more thorough investigations, it becomes apparent that they are practical designs of the PLA era, that the combinational circuits were extracted in the context of ancient work flows, or that their descriptions underwent massive transformations.

A great remedy would be to know how far can a circuit description deviate from its designer's source code, and in what directions. What is acceptable similarity, in the experiments mentioned above? Such knowledge is not available. In the further text, we are going to outline why it is so.

In [13] and [9], we conducted experimental studies to explain why the results in [7] and [12] were so poor. From their summary below, it will follow that the margin in allowable changes (transformations) in a description of a circuit is perhaps much smaller than practiced in current experimental evaluation. Lacking firm limits in those changes, one must be *prudent* in choosing and maintaining circuit examples with the intent to use them as benchmarks.

2 The class of practical combinational circuits

Experimental algorithmics [24][18] asks for a collection of instances for evaluation. One of the most straightforward methods to acquire such a collection is to *characterize* the class of permissible instances and then randomly *generate* a set of instances satisfying the characterization.

Such characterization seems not to be possible with circuits. The attempts to characterize *Boolean functions* used in engineering started with the use of Boolean algebra for logic synthesis [31]. The idea that practical functions are decomposable appears again in [25] and most notably in [27], where it is generalized to the conjecture that human design follows detectable patterns. Other properties such as group invariance [29] and information flow [22] have been tested, without decisive conclusions.

Boolean functions are only a part of the picture; a circuit description has *structure*, such as Boolean network. Iterative procedures depend on that structure, in often strange ways [26][32][10][15]. A provably practical Boolean function can thus have *impractical* descriptions, which a designer would not write down and the tools would not process effectively.

Both the papers [7] and [12] suggest to use altered descriptions of circuits as benchmarks. The former paper sees the complexity added to the circuit as a surrogate for redundancy introduced by a less-competent designer.

3 Experiments with structure loss

The examples in [7] and [12] have the original structure wiped out by collapsing the description into a Sum of Products (SOP) and forcing the tools to understand that SOP as a *structural* description. This is a special case of deriving an example by *transformation* that preserves the function of a circuit and produces an example with altered (suppressed) structure.

Already in [13] we documented that the classical decomposition-based synthesis, which starts with two-level minimization, provides much better results than the iterative tools used in the original studies. The only requirement was that the decomposition used is general enough to produce the design patterns required for acceptable solutions.

Iterative procedures gradually transform the description of a circuit. The intermediate descriptions can be seen as states of a local search procedure. Sub-par results then can be the results of trapping the search in local optima, which in turn can be improved by slower convergence and greater iterative power of the search. We realized this approach by randomization and repeating the procedure [14]. For some circuits, even an enormous investment of computing time did not ensure acceptable results.

During initial experiments, we noticed that, in some cases, the output circuit size is *correlated* with the size of the input circuit description. Randomizing the transformation used to suppress structure, we were able to produce different versions of input, and to characterize that dependency. We used 490 circuits from the following sources:

- LGSynth'93 benchmark set [23], 129 circuits; at the time, we were not aware of the manipulation of the set
- MCNC benchmark set [34], 103 circuits
- Altera OpenCores, converted to BLIF by Alan Mishchenko using Quartus II with standard settings, 110 circuits; may contain adders
- ISCAS'85 benchmark set [4], 11 circuits
- ISCAS'89 benchmark set [3], 50 circuits
- LGSynth'91 benchmark set [35], 2 circuits
- Illinois set, 7 circuits [6]
- LEKU examples by Cong and Minkovich, 8 circuits
- ITC'99 benchmark set [8], 22 circuits
- manually encoded n-bit adders, 16 circuits

In the first round of experiments, we produced a number of different SOPs transformed into structural description for each circuit, as in [7] and [12]. For 270 circuits, the number of SOPs was sufficiently large to provide reliable characterization, giving 7414 input descriptions (data points).

We processed the description by a number of synthesis tools, from which we report the results for ABC with the command sequence *dch,if,mfs* as typical for the set.

5801 cases (78%) belonged to circuits, where Spearman correlation between input size and output size was above 0.5. We further characterized the dependency by linear and convex nonlinear approximation in the logarithmic domain, which gave the values of *first derivative*. The value of 1 thus mean linear dependency, and lower values denote sublinear dependency.

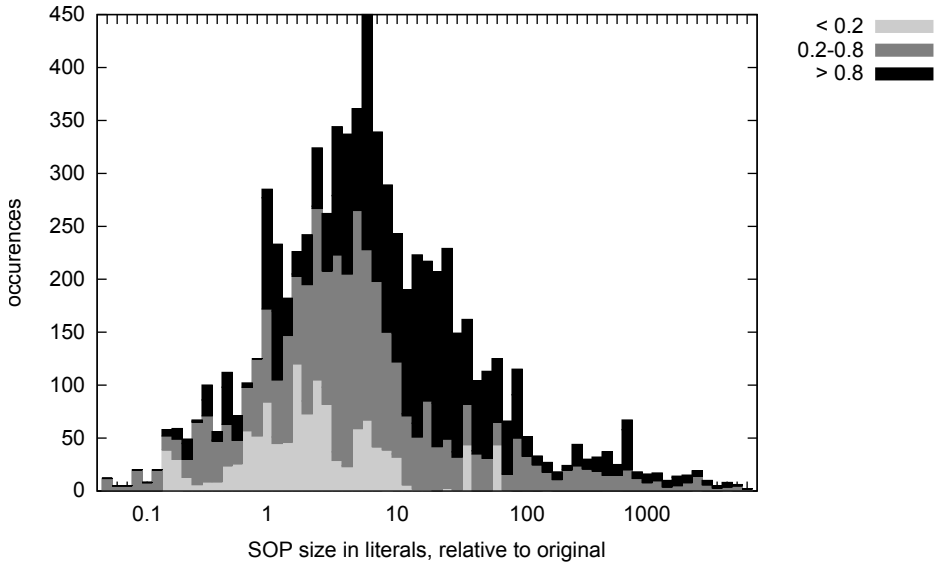

Figure 1: The impact of transforming circuit examples into SOP, and the derivative of the input size – output size dependency for *dch,if,mfs*

The situation can be seen in Figure 1. In 3097 cases (42%), the derivative is above 0.8. It indicates that the tool was unable to reach a reasonable solution; it just reduced the size by an (almost) constant factor. This cannot happen in normal use, or it would be noticed. Hence, synthesizing structural SOPs must be considered to be outside the intended operating region, and therefore not much relevant for tool evaluation.

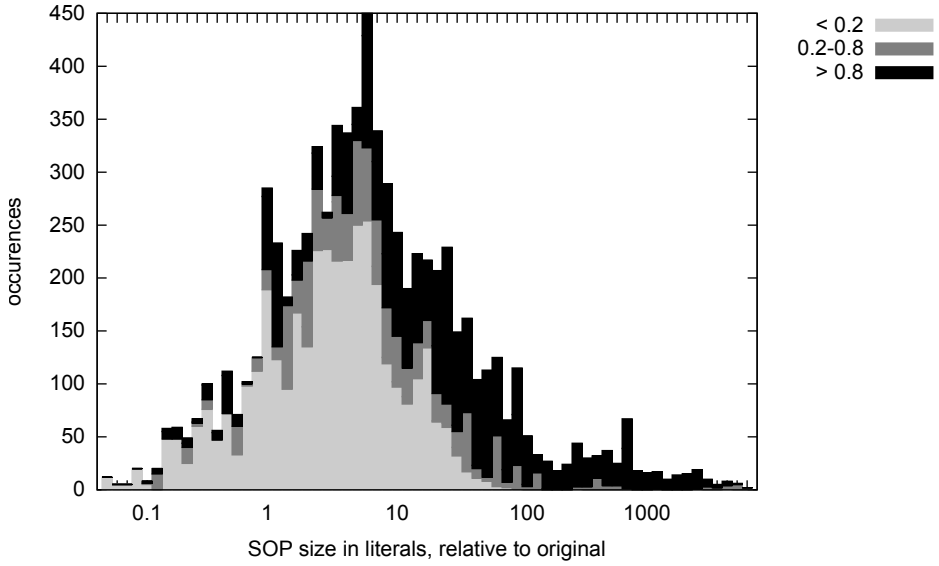

Figure 2: The impact of transforming circuit examples into SOP, and the derivative the input size – output size dependency for 20 iterations of *dch,if,mfs*

Figure 2 documents how iterative resynthesis [11] improves the situation. The number of cases with low derivative increased from 16% to 54%, while the number of cases with high derivative decreased from 42% to 29%.

In further experiments, we used a different kind of structure to replace the original one: a multiplexer structure derived from a Binary Decision Diagram (BDD). The results were similar in nature, with 55% of the cases exhibiting high derivative. The important fact is, that the corresponding derivatives from both transformation methods are *uncorrelated*. Spearman correlation is 0.044, and even 0.014 if we allow the approximations to extrapolate in order to have

more data points.

To conclude, the size of the solution was strongly correlated with the size of the input description. The slope of the dependency is given by

- the circuit,
- the synthesis procedure under test,
- the kind of replacement structure (SOP or multiplexers), and
- the size of the input itself.

The results can be interpreted to document that any procedure tolerates the loss of structure to some extent depending on other characteristics. Beyond that limit, it merely improves the circuit by a roughly constant factor.

For experimental evaluation, it follows that

- synthesis procedures have adapted in time to the nature of input, which is human design;
- various sources of redundancy are not interchangeable; genuine incompetent designer cannot be substituted by circuit transformation.

From both the sections above, it seems that the researcher's position with regard to benchmarks is much more limited than it seemed before. With our inability to characterize practical input, and with the tools' sensitivity to original structure, the only way seems to be *prudent*.

4 Loss of fidelity

When we become more prudent as dictated by the above outlined situation, several flaws appear in current benchmark sets and their treatment.

4.1 Context

Combinational synthesis is but a small part of a contemporary synthesis tool. The input that is genuine is the designer's HDL; between that and the input to logic synthesis procedure, a number of steps exist, which can work in different ways, depending on the following capabilities of the procedure:

- to work with hierarchical descriptions,
- to process completely independent parts of the circuit (components) effectively,
- to produce efficient implementation for specific circuits (e.g., adders), even if they are not marked as special,
- to do so even if the mapping from HDL constructs is "simple-minded", as in [20].

To have the HDL processing and elaboration under control, we need an open elaboration tool, such as [2] promises to be, and to document the derivation of actual synthesis examples from the original HDL code. Any evaluated synthesis procedure should state the context in which it is supposed to work.

4.2 Language of description and its semantics

The base notion behind circuit description in most academic tools is the Boolean network. This is also the case for combinational circuit described structurally in HDLs. With this in mind, it seems easy to translate descriptions between such languages. The difficulty, however, lies in node functions. Some languages, e.g., BENCH [5], have a built-in repertory of node functions. Others, e.g. the structural view in EDIF [1], do not describe node function at all; it must be guessed from the type name. In hierarchical HDLs, one has to decide which level of hierarchy describes node function and which have to be flattened. The ability to express incompletely defined function also varies. Hence, translation is not a trivial task, and in situation where mere reordering of declaration may change performance, cannot be considered prudent.

4.3 Instance processing and transformation

Certain benchmarks were processed by synthesis or simplification, with the belief that the tools provide ideal transformation without any loss of information. One such transformation occurred between the IWLS'91 [35] and IWLS'93 [23] sets. The aim was to use an industrial standard language (EDIF) instead of academic formats. The above outlined difficulty with language

semantics forced conversion from the large Boolean network nodes found in the former BLIF description into primitive gates. Where such large nodes did not occur, the tool preserved input complexity. In other cases, however, the size increased by an order of magnitude.

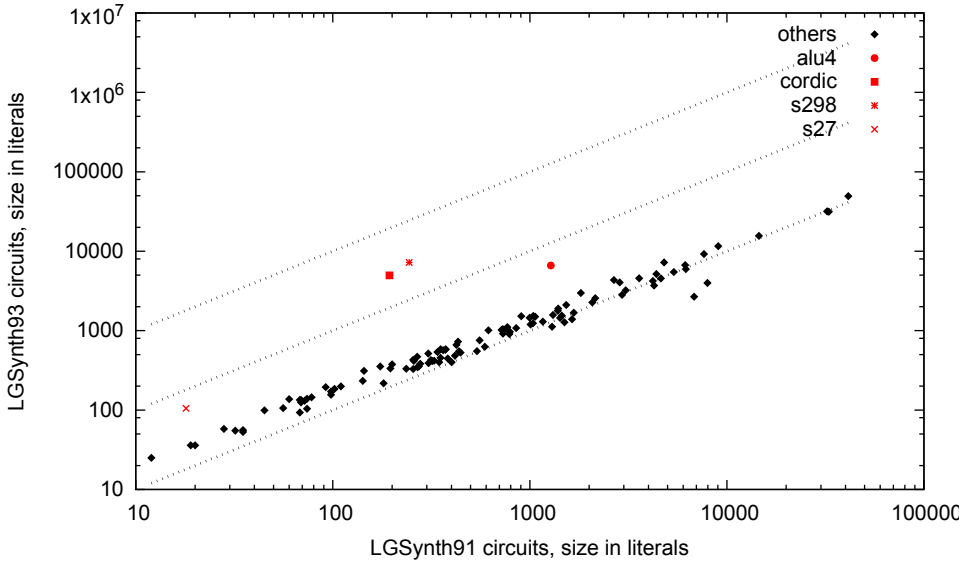

Figure 3: A comparison of circuits in the LGSynth91 and LGSynth93 sets. Dotted lines mark the 100:1, 10:1, and 1:1 size ratio.

Figure 3 shows the consequences of the manipulation. For majority of circuits, their size measured in literals was not much affected, with the exception of the four circuits highlighted in the graph. They were enlarged to the extent that they no longer represent the same benchmark, and we doubt if their structure is still representative for practical circuits. We were unable to replicate the effect of the manipulation even by old tools of the period. Practically all recent tools are unable to repair the harm in these cases, but usually are able to synthesize acceptable results from the original LGSynth91 circuit description.

We also studied the impact of the above mentioned manipulation to synthesis. Circuits from both sets were synthesized using the ABC system with the commands sequence *dch,if,lutpack* and are compared in Figure 4. The descriptions of circuits *s27* and *s298* contained Don't Care states in the form of a library cell named DC. These circuits could not have been synthesized. The problematic circuits *alu4* and *cordic* remained problematic, although the synthesis improved the ratio to LGSynth91. Surprisingly, new problems with circuits *frg1*, *too_large*, and *z4ml* appeared. It confirms that size alone is not sufficient to characterize the circuits. Furthermore, it forces us to mark the *entire* set as unreliable, because the altered structure of other circuits in the set can be difficult for other tools, and would not test the tools for anything that can be present in practical circuits.

We tried to provide difficult examples using non-standard work flow (collapsing, [12]). In the light of our later findings, this is not prudent. So is the hope of Cong and Minkovich [7], that the complexity introduced by collapsing may replace the overhead found in less competent designs.

[21] produces examples by application of random transformations to a circuit. The reported impact on SIS [30] performance is two- to four-fold. We still do not know if these examples test anything relevant.

In experimental algorithmics [24], examples may be generated randomly, within a class characterized by selected parameters. This cannot be done for circuits, and hence random generators using this approach [19] do not seem appropriate.

4.4 Documentation and optimum solutions

The function of a benchmark circuit has been mostly considered unimportant. Yet for many of the benchmarks, a high quality – possibly manual – solution did exist. Such a solution can be

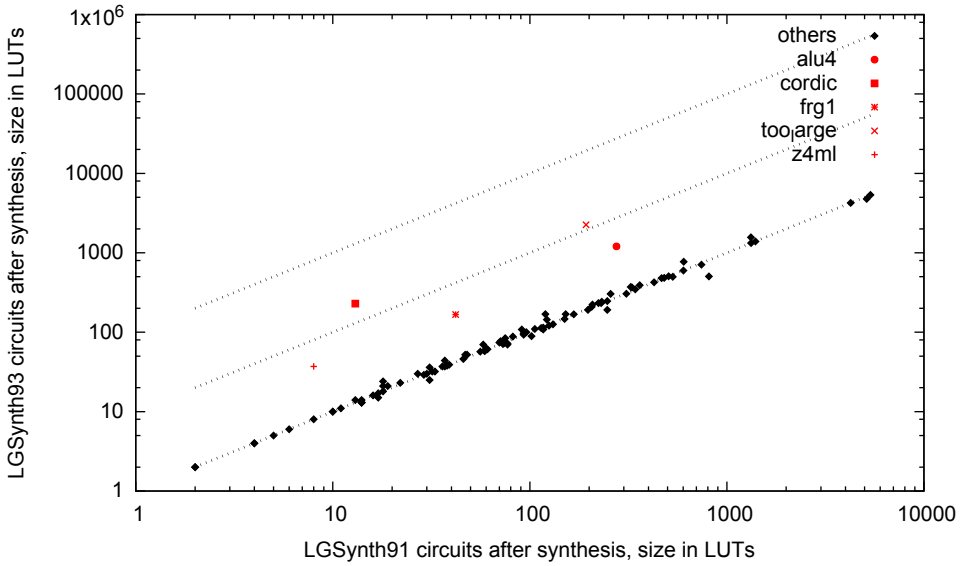

Figure 4: A comparison of circuits in the LGSynth91 and LGSynth93 sets *after synthesis*; *s27* and *s298* missing. Dotted lines mark the 100:1, 10:1, and 1:1 size ratio.

used as a valuable reality check.

As an example, take the famous *alu4* benchmark. For decades, it is known that the description is highly redundant [16]. The results provided by standard tools, having around 1000 gates, were considered normal. After a long effort, we verified that *alu4* is the 74181 circuit from Texas Instruments. A high quality manual solution is available since 1970 [33], which has the equivalent of 85 two-input gates. Had this been known, many researchers would have realized that their results are almost irrelevant.

A rare example of post factum documentation is [17]; it permits to make links between the broader class of an example and the tools' behavior.

4.5 Original purpose

So far, we wrote about experimental evaluation in general, with the primary purpose to prove performance improvement. In the process of tools development, however, examples can be used that are specially crafted in order to test specific situations or specific parts of the tools. One such example is the *ex1010* circuit, which circulates in various forms for a quarter of a century. Only in historical literature [28], we learn that this is a *synthetic* instance, generated *randomly*, and originally created to test two-level minimization. Even the original report finds that minimization procedures behave differently on real and synthetic examples, and tabulates them separately.

Similarly, examples for Automatic Test Generation (ATPG) [4] [3] may have been selected because they were difficult for ATPG tools. It is not clear how relevant they are for synthesis.

5 Conclusions

Intentional and unintentional experiments with loss of circuit structure teaches us to preserve circuits used for experimental evaluation as much as possible. Because we are unable to characterize the 'practical circuit', we cannot be sure which alteration – by synthesis, translation or random transformation – is harmless. Therefore the whole 'life' of an example must be documented, starting with the original HDL code and including any subsequent processing. Optimum solutions are valuable for experimental evaluation, so that they must be maintained.

Acknowledgment

This work was partially supported by the grant GA16-05179S of the Czech Grant Agency, "Fault Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features" (2016-2018).

References

[1] *ANSI/EIA-548-1988, Electronic Design Interchange Format, Version 2.0.0*. 1988.

[2] G. Bartsch et al. *ZamiaCAD: Open Source Platform for Advanced Hardware Design*. 2012. URL: `http://zamiacad.sourceforge.net`.

[3] F. Brglez, D. Bryan, and K. Kozminski. "Combinational profiles of sequential benchmark circuits". In: *Circuits and Systems, 1989., IEEE International Symposium on*. May 1989, 1929–1934 vol.3. DOI: `10.1109/ISCAS.1989.100747`.

[4] F. Brglez and H. Fujiwara. "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan". In: *Proceedings of the International Symposium on Circuits and Systems*. 1985, pp. 663–698.

[5] D. Bryant. *The ISCAS'85 benchmark circuits and netlist format*. 1988. URL: `http://www.facweb.iitkgp.ernet.in/~isg/TESTING/bench/iscas85.ps`.

[6] V. Chickermane, J. Lee, and J. Patel. "A comparative study of design for testability methods using high-level and gate-level descriptions". In: *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on*. Nov. 1992, pp. 620–624. DOI: `10.1109/ICCAD.1992.279302`.

[7] J. Cong and K. Minkovich. "Optimality Study of Logic Synthesis for LUT-Based FPGAs". In: *IEEE Trans. on Computer-Aided Design* 26.2 (Feb. 2007), pp. 230–239.

[8] F. Corno, M. S. Reorda, and G. Squillero. "RT-level ITC'99 benchmarks and first ATPG results". In: *Design Test of Computers, IEEE* 17.3 (July 2000), pp. 44–53. ISSN: 0740-7475. DOI: `10.1109/54.867894`.

[9] P. Fišer and J. Schmidt. "A Difficult Example Or a Badly Represented One?" In: *Proc. of 10th International Workshop on Boolean Problems*. Freiberg, DE: Technische Universität Bergakademie, 2012, pp. 115–122. ISBN: 978-3-86012-438-3.

[10] P. Fišer and J. Schmidt. "How Much Randomness Makes a Tool Randomized?" In: *Proc. of the 20th International Workshop on Logic and Synthesis (IWLS)*. San Diego, California, USA, 2011, pp. 136–143.

[11] P. Fišer and J. Schmidt. "Improving the iterative power of resynthesis". In: *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2012 IEEE 15th International Symposium on*. Apr. 2012, pp. 30–33. DOI: `10.1109/DDECS.2012.6219019`.

[12] P. Fišer and J. Schmidt. "Small but Nasty Logic Synthesis Examples". In: *Proc. of 8th Int. Workshop on Boolean Problems (IWSBP)*. Freiberg, DE: Technische Universität Bergakademie, 2008, pp. 183–190. ISBN: 978-3-86012-346-1.

[13] P. Fišer and J. Schmidt. "The Observed Role of Structure in Logic Synthesis Examples". In: *Proceedings of the International Workshop on Logic and Synthesis 2009*. Berkeley, CA, USA, 2009, pp. 210–213.

[14] P. Fišer, J. Schmidt, and J. Balcárek. "On Robustness of EDA Tools". In: *Proceedings 17th Euromicro Conference on Digital Systems Design (DSD)*. Verona (Italy), 2014, pp. 427–434.

[15] P. Fišer, J. Schmidt, and J. Balcárek. "Sources of Bias in EDA Tools and Its Influence". In: *Proc. of 17th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS)*. Warsaw, Poland, 2014, pp. 258–261.

[16] M. Fujita et al. "Multi-level Minimization by Partitioning". In: *Logic Synthesis and Optimization*. Ed. by T. Sasao. Springer Science & Business Media, 1993, pp. 109–126.

[17] M. C. Hansen, H. Yalcin, and J. P. Hayes. “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering”. In: *IEEE Design & Test of Computers* 16.3 (1999), pp. 72–80. ISSN: 0740-7475. DOI: http://doi.ieeecomputersociety.org/10.1109/54.785838.

[18] H. H. Hoos and T. Stützle. “Empirical analysis of randomized algorithms”. In: *Handbook of Approximation algorithms and metaheuristics*. Ed. by T. F. Gonzalez. Boca Raton (FL): Chapman & Hall, 2007, pp. 14.1–14.17.

[19] M. Hutton et al. “Characterization and Parameterized Generation of Synthetic Combinational Benchmark Circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17.10 (Nov. 1997), pp. 985–996.

[20] Integrated Systems Laboratory LSI. *The EPFL Combinational Benchmark Suite*. 2016. URL: http://lsi.epfl.ch/benchmarks.

[21] K. Iwama and K. Hino. “Random generation of test instances for logic optimizers”. In: *31st ACM/IEEE Design Automation Conference*. San Diego, California, United States: ACM, 1994, pp. 430–434.

[22] L. Jóźwiak. “Information relationships and measures: an analysis apparatus for efficient information system synthesis”. In: *EUROMICRO 97. New Frontiers of Information Technology., Proceedings of the 23rd EUROMICRO Conference*. Sept. 1997, pp. 13–23. DOI: 10.1109/EURMIC.1997.617209.

[23] K. McElvain. *LGSynth93 benchmark set: Version 4.0*. 1993.

[24] B. Moret. “Towards a discipline of experimental algorithmics”. In: *Data Structures, Near Neighbor Searches, and Methodology: Fifth and sixth DIMACS implementation challenges*. Ed. by M. Goldwasser, D. Johnson, and C. McGeoch. AMS, 2002, p. 15.

[25] M. A. Perkowski and S. Grygiel. *A Survey of Literature on Function Decomposition – Version IV*. Study Report. Portland State University, 1995.

[26] A. Puggelli et al. “Are Logic Synthesis Tools Robust?” In: *Proc. of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2011, pp. 633–638.

[27] T. D. Ross et al. *Pattern Theory: An Engineering Paradigm for Algorithm Design*. Final Report ADA243214. 1991.

[28] R. L. Rudell. *Multiple-Valued Logic Minimization for PLA Synthesis*. Tech. rep. M86/65. University of California in Berkeley, ERL, June 1986.

[29] Y. L. Sagalovich. “Mera uporjadochenostii bulevoj funkcii”. Russian. In: *Problemy Peredachi Informatsii* 8 (1961), pp. 88–108.

[30] E. Sentovitch and K. S. et al. *SIS: A System for sequential circuit synthesis*. Tech. rep. M92/41. May 1992.

[31] C. E. Shannon. “The synthesis of two-terminal switching circuits”. In: *Bell Systems Technical Journal* 28 (1949), pp. 59–98.

[32] W. Shum and J. H. Anderson. “Analyzing and predicting the impact of CAD algorithm noise on FPGA speed performance and power”. In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA ’12)*. 2012, pp. 107–110.

[33] Texas Instruments Inc. “Type SN74181 Aritthmetic Logic Unit/Function Generator”. In: *TTL Catalog Supplement*. 1970, S7–1.

[34] S. Yang. *Logic Synthesis and Optimization Benchmarks User Guide*. Technical Report 1991-IWLS-UG-Saeyang. Research Triangle Park, NC: MCNC, Jan. 1991.

[35] S. Yang. *Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0*. Tech. rep. MCNC Technical Report, Jan. 1991.