Information and Guidance for Class Projects

ECE/CS 5745/6745: Testing and Verification of Digital Circuits

Priyank Kalla
ECE Dept., Univ. of Utah
kalla@ece.utah.edu

Latest Update: December 12, 2023. This document is always a work in progress. I will continue to update it with more project options as and when I get more ideas.

I. Introduction

Students in the class are required to work on a term project. You will work on the project as a team of 2 or 3 students. The project should address hardware or software[1] verification, core computational technologies for verification (SAT/SMT-solvers, Gröbner Bases, extensions of Binary Decision Diagrams (BDDs), AIGs, etc.), or consider other applications of verification technologies – such as hardware security, logic encryption, etc. Verification applications may range from combinational circuit verification, sequential circuit verification, RTL datapath verification (Verilog, C-programs), to any problem that can be formulated as such. The core methods should include concepts from testing and verification, such as algebraic geometry and symbolic computation (Gröbner basis reasoning), SAT-based function abstraction, post-verification debugging (bug location and characterization) of incorrect circuits, but not necessarily limited to it. Your formulations can be a mixture of algebraic geometry, symbolic computation, SAT, AIGs and BDDs. Since computational algebraic geometry is a core component of the course, studying some advanced Gröbner basis algorithms – or tweaking them for application to circuits or SAT solving – is also allowed.

The project will require that you develop a simplified CAD framework (i.e. do some coding or tool integration, etc.) for test and verification, conduct some experiments with various tools, gain some experience about how verification techniques are applied on practical circuits/systems. You may also develop core verification engines (solvers), and benchmark their execution on available circuits/designs. It is also possible to formulate your project around a theoretical investigation. The choice of the project is yours — remember, however, that I am the boss, and I will have the final say about your project deliverables ☺.

II. Time-line

Project time-lines are as follows:

1) Form a team of 2-3 members. Feel free to use the Canvas discussion board to scout for team members.
2) Schedule a first meeting with me either this week or early next week. [I will send you my availability after I setup a Doodle-like poll for you to pick a time.]
3) We will discuss your interests and your project options, and I may help you with some reading materials, tools, benchmarks, etc.

[1] Acceptable only if someone from the CS side has a good ideal of what software verification problem (s)he wants to address, and which software/infrastructure tools are available to use for the project

4) By Nov 15, your project should be finalized. you will submit a project title, team members info, and 1-2 page(s) abstract/summary of your project proposal. 1 submission per team.
5) **The final project report, and any tools developed, will be due soon after exams end, i.e. by Monday December 18.** Details on project submission will be sent out later.
6) We can have further project meetings and consultations as and when required.

III. Project Options

In this section, I will provide you with some suggestions on what kind of projects you can work on. I am also providing you with some references, which can be found on IEEEXplore or ACM Digital Library. From the Univ. of Utah network, you have free access to almost all of these papers. In case you cannot find these references, contact the TA or me. All papers authored by me are available on my website too.

A. Verification Core Technologies

Core technologies for verification include SAT solvers, decision diagrams such as BDDs and ZDDs, SMT-solvers, and various Symbolic Computing engines such as Gröbner basis engines, number-theoretic solvers, etc. You will notice that solvers – along with their source code – are already available to us. Therefore, just re-designing/re-writing the same solver engines is probably not a fun exercise. However, building extensions on these base engines is acceptable – but is **not** my favourite option.

B. Developing an ATPG Tool – updated December 12, 2023

Those who liked BDDs or AIGs, can actually develop a BDD/AIG-based fully automatic stuck-at-fault ATPG tool. Given a blif file, you can identify all the check-points of the circuit, and then generate tests for checkpoint faults.

There is a very interesting paper from Prof. Fujita and Alan Mishchenko regarding *multi-fault ATPG* [1]. It used a SAT and Partial Synthesis style approach to multiple fault ATPG. A simplified study of this paper might also be worthwhile. I am also interested in studying this topic for arithmetic circuits, and I wonder how SAT fares on arithmetic circuits for multi-fault ATPG. Would there exist a Gröbner basis analog for this problem? A semi-theoretical study with some simple experimentation on a couple of circuits might be a worthwhile class project.

C. Gröbner Bases and SAT

Solving SAT problems using Gröbner bases (GB) is a very fundamental problem, it does have a rich history, but not rich enough to the liking of most SAT-researchers. The analog of Boolean SAT in the GB-domain is to use Buchberger's algorithm over $\mathbb{F}_2[x_1, \ldots, x_n]$. A good start is my paper [2], where you may find references to many other SAT+GB frameworks. I can also provide you with Condrat's M.S. thesis on this same topic.

GB computation is very susceptible to the term ordering used in Buchberger's algorithm. SAT is also quite dependent on the various decision heuristics used for solving. You could study and learn about constraint partitioning (by way of hypergraphs) and variable ordering from the SAT domain, and apply these techniques to derive efficient term ordering for GB engines. Again, a good start is my paper [3], and it's extensions [4].

One of the most important contribution recently in SAT is about conflict clauses. Finding conflict clauses is akin to information mining from constraints. Buchberger's S-polynomials in the Gröbner basis algorithm also "mine" more information about ideals, in terms of polynomials with new leading terms. *What relationships do S-polynomials have with conflict clauses?* This problem is not well understood, so it might be a good idea to study this problem. A relevant paper is [5].

In class, we have also studied about UNSAT cores of CNF formulas. *Can the Gröbner basis algorithm be used to identify UNSAT cores?* You could try modifications to the techniques presented in my paper [6]. On similar lines, one can also solve #-SAT, i.e. count the number of SAT solutions. See Gao's M.S. thesis [7].

UNSAT cores are a type of a "minimum UNSAT (min-UNSAT)" problem. Its dual is MAXSAT. [Do a search on the internet on the MAXSAT problem, and contact me for some tutorial slides]. Wouldn't you be interested in finding the Gröbner basis analog of the MAXSAT problem, or finding UNSAT cores of polynomial ideals?

D. On Gröbner Basis Engines

Buchberger's algorithm is old, circa 1963. In the last couple of decades, a new algorithm has been devised by J.-C. Faugére called the F_4 algorithm [8]. It is based on the same principles as Buchberger's algorithm (reduce all $Spoly(f, g) \xrightarrow{G}_+ 0$). However, it performs a simultaneous reduction of many S-polynomials in one-go, based on a Gaussian-elimination Matrix reduction. A team may actually study and implement the F_4 algorithm. *Or you may find it more feasible to implement polynomial reduction in the style of* F_4 – *i.e. implement polynomial division using a matrix based Gaussian elimination/row reduction.* It is a very do-able project, may also turn out to be kind of easy. If you are interested, talk to me and we can probably even think of a few extensions of this algorithm and/or tweak it for circuit verification. The original F_4 paper is going to be very hard to read. However, there is a nice M.S. thesis [9], also available on the web (I have a copy), that describes F_4 in Chapters 1 and 2. Prof. Cox's book [10] (its latest 4th edition) has a description of a simplified version of the F_4 algorithm.

After F_4, Faugére came up with it's extension called the F_5 algorithm for Gröbner basis. That algorithm can also be a class project. Simplified versions of these algorithms can be implemented within Singular, where you can make use of Singular's data-structures and routines to implement the algorithms using Singular's scripting language. For Fall 2023, this might be a bit too difficult, though, as F_5 needs a better understanding of Gröbner bases.

E. Boolean Gröbner Bases

Since we all want to exploit the power of GB engines for Boolean circuits, there have been attempts to build Boolean Gröbner Basis engines for model checking and SAT solving. This could be a good project as it combines a decision-diagram, i.e. a BDD-like DAG representation (Zero-Suppressed BDDs or ZDDs) and GB engines [11] – so you can learn both BDD-type unique tables and Gröbner Bases. What's cool is that the ZDD library is available in CUDD, so you should already be familiar with it. This project requires representing Boolean polynomials with BDDs, and modeling arithmetic operations $\pmod 2$ as AND-XOR operations on BDDs – it's pretty cool. You can refer the following papers by Utkarsh [12], [13]. This could be a good project for those of you who enjoyed programming in the BDD package.

If one can use BDDs for (AND/XOR) polynomial manipulation, then why not use AIGs instead? That could be another option: *Boolean Gröbner basis computation on digital circuits using AIGs*. Since the ring $\mathbb{F}_2 \equiv \mathrm{B}$, we can model the polynomial division algorithms over $\mathbb{F}_2$ using AIG rewriting. This means that sets of (Boolean) polynomials can be represented using AIGs – and vice-versa – and that polynomial division or GB computations can be implemented in ABC. The paper [12] can give you some inspirations in this regard. Other similar ideas include [14] [15], but might be more work. *Those of you who really liked experiments with the ABC tool, you might be inclined to explore how to program using the AIG library, and use the Boolean manipulation techniques to mimic polynomial division.*

F. Combinational Circuit Verification

Recent work has seen very successful application of GB engines to combinational circuit verification. The approach can be applied over Galois fields $\mathbb{F}_{2^k}$ or over integer arithmetic circuits. However, the GB computation over large circuits can be computationally infeasible due to the complexity of the GB algorithm. There have been many investigations on how this complexity can be avoided, with some very cool results in the last 3 years.

One paper that completely obviates the need to compute a GB is my own [16], download the extended version from my website. [This paper is very easy to read and is written in a way that an average computer engineer can understand, with good tutorial content.] Then there is another paper that solves the reverse problem: Given a circuit C, identify the function implemented by it, by efficient application of Gröbner Basis computations [17] [18]. Study/extensions of these techniques is also going to be a very good project.

Update December 12, 2023: **GPU computing:** The papers [16] [17] perform GB computations, but rely on a matrix-based representation of the problem. I have always been looking for someone to try out versions of GB-based combinational circuit verification using GPU computing. Languages such as CUDA have a lot of matrix-based support, and polynomials derived from circuits have sparse matrix representations. I have always wondered if performing GB-based reductions for verification can benefit from GPU computing? One attempt could be to try the technique of [16] on a GPU.

G. Verification of Integer Multipliers

While the aforementioned papers [16] [17] solve verification problems over Galois field circuits, there are *very similar applications* over integer arithmetic circuits. In the past couple of years, researchers in industry and academia around the world have also solved the verification of integer multipliers. This was considered a very hard problem until recently, almost unsolvable automatically without manual intervention, but is now pretty much solved due to the recent papers: i) a paper published in [19] which you can download from here: http://fmv.jku.at/papers/RitircBiereKauers-FMCAD17.pdf. This paper won the best paper award at FMCAD! It's latest avatar, along with their developed verification engine (AMulet) is [20]. The tool can be obtained from http://fmv.jku.at/amulet2/. While we will study the basic ideas of [19] in class, a detailed study of this paper can be a class project. *Does Amulet/integer arithmetic verification have any scope for speed-up via GPU computing?*

Parallel to `AMulet's` development, there has been a similar effort from Prof. Rolf Drechsler's group for verification of integer multiplier circuits, leading to the development of the REVSCA tool [21]. From [21],

you can find references to their previous papers in 2018 and 2016. You could try to implement simplified versions of these papers in Singular, or ABC, or even python, to learn how they take textbook concepts and apply them to solve practical problems.

H. SMT-solving for Word-Level RTL datapaths using non-GB engines

Update December 12, 2023: This topic remains very hard/infeasible to solve practical problems. I would not strongly recommend this study anymore.

GB can *help in solving* a system of polynomial equations, but it *cannot actually solve* a system of polynomial equations. Therefore, some sort of a SAT/SMT or a number theoretic solver is still needed. One way to solve the RTL datapath verification problem is using *Wedler et al* [22]. Another orthogonal approach is based on Newton's p-adic iteration [23]. Probably, a sweet spot is a mixture of both number-theoretic and GB-based approaches.

If you are interested in working over RTL-datapaths, and don't mind exploring a slightly more *investigative* project, talk to me about it, and we could explore if *bit-level* techniques such as [24] can be explored over word-level RTLs.

I. Sequential Circuit Verification

If you are interested in sequential circuit verification, then the first thing to do is to study the sequential circuit verification chapter from the textbook [25] (available in our library), and understand how BDDs are used for implicit state enumeration of sequential circuits (we will study these in the class too, but only in the last couple of lectures). Then, you can explore GB engines to do just that — implicit state enumeration. Two papers that address such issues [26] [27].

If you have taken courses in model checking, then there is a world of knowledge out there that applies GB engines for model checking [15] [14].

IV. Post-Verification Debugging and Auto-correction of Arithmetic Circuits – Updated December 12, 2023.

Consider a combination circuit C, which is correctly designed. Assume that all the gates in the circuit are known, except only 1-gate is unknown. You are asked to identify the function implemented by that gate. This is a problem of **rectification** that we have studied in class. The corresponding papers are [28] that used Craig interpolation, and another which uses iterative SAT solving [29]. One of the projects could be to use these ideas and recreate his experiments using a SAT solver. Or you could even implement these by programming within ABC.

Another option would be to use the Gröbner basis engine instead of a SAT solver for *single-fix rectification* of arithmetic circuits. My students have recently studied these problems using: i) Craig interpolation [30], for which I have a copy of an extended book chapter that I wrote for it. ii) using extended ideal membership testing [31]. So maybe you could study these papers too and implement them in Singular.

This problem formulation of [29] actually inspires the post-verification debugging problem: Given a buggy circuit C, let X denote the set of nets/wires of C. Can you rectify/fix this circuit at some m-target nets $W = \{w_1, \ldots, w_m\} \subset X$? This is called *multi-fix rectification*, which requires identifying potential (or

potentially "good") m rectification targets, and then to compute corresponding rectification functions at each of these m targets. The rectification techniques in [32], [33] iteratively and incrementally compute multiple single-fix functions that partially patch the circuit in each iteration. The more recent techniques further include more resource awareness in patch generation by reusing existing logic [34], employing improved heuristics for target selection [35], or enumerate rectification points and their drivers by a combination of simulation and re-wiring strategies [36]. Interested readers may refer to [37] for a comprehensive review of rectification techniques.

In recent years, the EDA industry has run design competitions to solve this rectification problem, as it is widely encountered in industry. Selection of good candidate nets W_m for rectification has recently been addressed in [35] [36]. *One project option for two 2-member teams, or one 4 member team, could be to explore signal selection for rectification (2 members) in given circuits, and to compute rectification patches (2 members). Also, the above techniques were designed for random logic circuits. How well do they work for integer arithmetic circuits? Can signal selection for rectification be improved for integer arithmetic circuits?*

For arithmetic circuits, theory and techniques for multi-fix rectification using Gröbner basis have recently been published by my students – [38] for finite field circuits and [39] for integer arithmetic circuits. These papers are available on my website. But these techniques [38] [39] assume that the target nets are given a priori. For arithmetic circuits, do we need some modifications to the techniques of [35] [36] to select better rectification targets? One could perform some simple experiments to get a rough idea about these issues.

V. Craig Interpolation in Algebraic Geometry

Just like Craig interpolation exists over Boolean (un)SAT problems, similarly, polynomial algebra over finite fields also admits Craig interpolation. We have built an entire theory of Craig interpolation in finite fields. We will look at it in the class too, using Gröbner basis with LEX term orders. ABC computes interpolation using a heuristic on UNSAT proofs. I am interested in finding analogues of UNSAT proofs in algebraic geometry and efficiently computing an interpolant. If you have any interest in these ideas, let me know and I will email you a manuscript that I wrote that sketches these ideas – and it has examples too!

VI. Applications in Hardware Security – Updated 11/05/17

In hardware security, SAT-solvers and ATPG engines are used very often for security validation, for attacks on hardware ICs to break obfuscation or logic locking. As Gröbner bases complement SAT solving, I am very interested in studying some of these topics by replacing a GB engine for a SAT solver and asking questions such as: *Can GB engines provide a mechanism or a metric to evaluate (or measure) hardware security provided by a technique? Can GB engines help derive a formal model of hardware security? Does classification of a variety (corresponding to the number of input vectors applied to a circuit) give us more information about security/logic encryption implemented to secure a circuit than what is provided by a SAT solver?* These type of projects can be very investigative, and a learning experience, as it involves modeling the problems by trial-experimentation-error-repeat, and I'll be interested in them.

An interesting application in hardware security is to crack-open a logic obfuscation or IC-camouflaging. A recent paper [40] uses a SAT solver to reverse engineer camouflaged gates in an obfuscated circuit. You could try the same experiments as given in this paper, or try to come up with a Gröbner basis formulation for

the same problem. A Gröbner basis formulation definitely exists, and relates to the concepts of projections of varieties; it can be solved by computing a Gröbner basis with a specific LEX term order!

Or you could try to study how hardware security based circuit designers are securing crypto-circuits to post-quantum crypto attacks. If you are interested in studying basic concepts in this domain, let me know. I'm currently building a database of these papers, and we can do this type of project together – well, sort of together ☺.

VII. Any other ideas you have?

Such as computing Gröbner bases for Neural networks, like binarized NN, or CNNs, etc., for verification? Any idea that you may have? Talk to me about it, and we'll see if it makes sense for a class project.

VIII. My Availability for Project Meeting

Let me send you a Doodle-like poll. If those times don't work for you, email me and we'll find some time.

IX. Project Proposal Submission Guidelines

Each group of students needs to submit a project proposal by Nov 15. *But before that, every team needs to meet with me at least once.* The proposal should be a 1-2 page(s) document and be uploaded on Canvas. It should describe:

1) A tentative title of your project and the constituent team members.
2) The general topic area that you plan to study, and the core computational technologies that you wish to employ. For example, are you interested in combinational circuit verification, sequential circuit or RTL verification? Will you learn Gröbner basis concepts and their application or will you use BDDs, SMT solvers, SAT solvers or a combination of these?
3) If you are interested in developing a core engine, say, the F_4 algorithm and applying it to a specific domain — such as over Boolean rings for bit-level circuit verification — then describe the objective clearly.
4) If you have already refined/finalized the exact problem you wish to solve, then state the problem as best as you can. If you only have a general idea of the problem, and plan to refine the exact problem later with my help, it is okay to say so. But, do you best to narrow down the focus as best as you can.
5) Once the problem is described, what approach will you take to address your project. What topic areas require further study and which papers will you refer to? Provide a (short/relevant) reference list.

Do not worry if your proposal is not a very precise and mathematically clean document for now. Give it your best shot. I will help you finesse it, and I am expecting that many of you will have to go through one more iteration with my feedback to refine it. Good luck! ☺.

X. Final Project Submission Guidelines – updated Dec 12, 2023

Your final project should be submitted by **Monday December 18, 2023, by midnight**. The project should be submitted on Canvas.

Your project submission should be a tar'ed and gzipped (`yournames.tar.gz`) or zipped (.ZIP) archive. The archive should contain the following:

- Of course, you will have a PDF document describing your project report. While I don't wish to impose a format on you, your manuscript should likely contain:

  - A project summary (abstract), a section on Introduction to your project, including maybe the specific Problem Statement. Include proper references. Be brief and to-the-point.
  - You may include a small section on Preliminaries or Notation. [E.g., "we denote the Boolean domain with $\mathbb{B}$, whereas $\mathbb{Q}[x]$ denotes the polynomial rings ...", and so on.]
  - A section describing your contributions, your approach to learning the concepts, the algorithms or techniques that you have studied, and you should demonstrate parts of your main contribution/study using example circuits, designs or code fragments.
  - If you have implemented any techniques in software, provide a pseudo-code. If you have run some experiments using your tools, provide some data in a results table or a graph.
  - You should include a section briefly describing all the concepts that you learned through the project. Please write this section carefully.
  - Division of labour: Honestly describe each team members contribution.
  - A final Conclusion section.
- If you have developed some software, include it in a sub-directory in your archive. If your software algorithms operate on a circuit (or other data) include them in this directory too.
- Or if you used Singular to run experiments, include those `*.sing` files.
- No need to include existing/public domain tools such as Singular, Gurobi, PicoSAT, ABC, etc. Include only the scripts that you developed, such as your Perl/Python scripts, or Singular Code. If you worked on ROBDDs, or ZBDDs, and your code spans 2-3 C/C++ files, include those files along with instructions on how to compile them with CUDD-version.
- A README file should be included describing your archive, or contain instruction on how to run your software on a circuit/problem.

Please format and write your report properly, using appropriate fonts, notations, etc. Try not to be verbose, but precise use of mathematical notations should be used. Use of LaTeXis not required, but you might find it preferable to use it for math-formatting.

Finally, proof-read and spell-check your document before submitting. Thanks and good luck.

References

[1] M. Fujita and A. Mishchenko, "Efficient SAT-based ATPG for all Multiple Stuck-at Faults," in *IEEE Intl. Test Conference (ITC)*, 2014.

[2] C. Condrat and P. Kalla, "A Gröbner Basis Approach to CNF formulae Preprocessing," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2007, pp. 618–631.

[3] V. Durairaj and P. Kalla, "Guiding CNF-SAT Search via Efficient Constraint Partitioning," in *Intl. Conf. on CAD (ICCAD)*, 2004.

[4] ——, "Variable Ordering for Efficient SAT Search by Analyzing Constraint-Variable Dependencies," in *Intl. Conf. on Theory and Applications of SAT Testing*, ser. LNCS, vol. 3569, 2005, pp. 415–422.

[5] C. Zengler and W. Küchlin, "Extending Clause Learning of SAT Solvers with Boolean Gröbner Bases," in *Intl. Workshop Computer Algebra in Scientific Computation*, 2010, pp. 293–302.

[6] X. Sun, I. Ilioaea, P. Kalla, and F. Enescu, "Finding Unsatisfiable Cores of a Set of Polynomials using the Gröbner Basis Algorithm," in *Intl. Conf. Principles and Practice of Constraint Programming*, 2016.

[7] S. Gao, "Counting Zeros over Finite Fields with Gröbner Bases," Master's thesis, Carnegie Mellon University, 2009.

[8] J.-C. Faugere, "A New Efficient Algorithm for Computing Gröbner Bases (F_4)," *Journal of Pure and Applied Algebra*, vol. 139, pp. 61–88, June 1999.

[9] M. Yu, "An f4-style involutive basis algorithm," Master's thesis, Univ. of Southern Mississipi, 2010.

[10] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 2007.

[11] M. Brickenstein and A. Dreyer, "Polybori: A Framework for Gröbner Basis Computations with Boolean Polynomials," *Journal of Symbolic Computation*, vol. 44, no. 9, pp. 1326–1345, September 2009.

[12] U. Gupta, P. Kalla, and V. Rao, "Boolean Gröbner Basis Reductions on Datapath Circuits Using the Unate Cube Set Algebra," *IEEE Trans. CAD*, vol. 38, no. 9, pp. 576–588, March 2018.

[13] ——, "Boolean Gröbner Basis Reductions on Datapath Circuits Using the Unate Cube Set Algebra," in *26th International Workshop on Logic & Synthesis*, pp. 124–131.

[14] M. Y. Vardi and Q. Tran, "Groebner Bases Computation in Boolean Rings for Symbolic Model Checking," in *IASTED*, 2007.

[15] G. Avrunin, "Symbolic Model Checking using Algebraic Geometry," in *Computer Aided Verification Conference*, 1996, pp. 26–37.

[16] J. Lv, P. Kalla, and F. Enescu, "Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits," in *IEEE Trans. on CAD*, vol. 32, no. 9, 2013, pp. 1409–1420.

[17] T. Pruss, P. Kalla, and F. Enescu, "Efficient Symbolic Computation for Word-Level Abstraction from Combinational Circuits for Verification over Finite Fields," *IEEE Trans. on CAD*, 2016.

[18] ——, "Equivalence Verification of Large Galois Field Arithmetic Circuits using Word-Level Abstraction via Groebner Bases," in *Design Automation Conf.*, 2014.

[19] D. Ritric, A. Biere, and M. Kauers, "Column-Wise Verification of Multipliers Using Computer Algebra," in *FMCAD*, 2017.

[20] D. Kaufmann and A. Biere, "AMulet 2.0 for Verifying Multiplier Circuits," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 12652, pp. 357 – 364, 2021.

[21] A. Mahzoon, D. Große, and R. Drechsler, "Revsca: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers," in *Design Automation Conference*, 2019.

[22] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G.-M. Greuel, "STABLE: A New QBF-BV SMT Solver for Hard Verification Problems Combining Boolean Reasoning with Computer Algebra," in *IEEE Design, Automation and Test in Europe Conference*, 2011, pp. 155–160.

[23] N. Tew, P. Kalla, N. Shekhar, and S. Gopalakrishnan, "Verification of Arithmetic Datapaths using Polynomial Function Models and Congruence Solving," in *Proc. Intl. Conf. on Computer-Aided Design (ICCAD)*, 2008, pp. 122–128.

[24] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths," in *Computer Aided Verification Conference*, 2008, pp. 473–486.

[25] G. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[26] X. Sun, P. Kalla, and F. Enescu, "Word-level Traversal of Finite State Machines using Algebraic Geometry," in *Proc. High-Level Design Validation and Test*, 2016.

[27] X. Sun, P. Kalla, T. Pruss, and F. Enescu, "Formal verification of sequential Galois field arithmetic circuits using algebraic geometry," in *Design Automation and Test in Europe, DATE 2015. Proceedings*. IEEE/ACM, 2015.

[28] K.-F. Tang, C.-A. Wu, P.-K. Huang, and C.-Y. R. Huang, "Interpolation based incremental ECO Synthesis for Multi-Error Logic Rectification," in *Proc. Design Automation Conf.*, 2011, pp. 146–151.

[29] M. Fujita, "Toward Unification of Synthesis and Verification in Topologically Constrained Logic Design," *Proceedings of the IEEE*, 2015.

[30] U. Gupta, I. Ilioaea, V. Rao, A. Srinath, P. Kalla, and F. Enescu, "On the rectifiability of arithmetic circuits using craig interpolants in finite fields," in *IFIP/IEEE Intl. Conf. VLSI (VLSI-SoC)*, 2018, pp. 1–6.

[31] V. Rao, U. Gupta, I. Ilioaea, A. Srinath, P. Kalla, and F. Enescu, "Post-Verification Debugging and Rectification of Finite-Field Arithmetic Circuits using Computer Algebra Techniques," in *Proc. Formal Methods in CAD*, 2018, pp. 121–129.

[32] K. F. Tang, P. K. Huang, C. N. Chou, and C. Y. Huang, "Multi-patch Generation for Multi-error Logic Rectification by Interpolation with Cofactor Reduction," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 1567–1572.

[33] H. T. Zhang and J. H. R. Jiang, "Cost-aware Patch Generation for Multi-target Function Rectification of Engineering Change Orders," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[34] A. Q. Dao, N.-Z. Lee, L.-C. Chen, M. P.-H. Lin, J.-H. R. Jiang, A. Mishchenko, and R. Brayton, "Efficient Computation of ECO Patch Functions," in *Design Auto. Conf. (DAC)*. ACM, 2018, pp. 51:1–51:6.

[35] Y. Kimura, A. M. Gharehbaghi, and M. Fujita, "Signal Selection Methods for Efficient Multi-Target Correction," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[36] V. N. Kravets, N. Lee, and J. R. Jiang, "Comprehensive Search for ECO Rectification Using Symbolic Sampling," in *Design Automation Conference (DAC)*, 2019, pp. 1–6.

[37] J. H. R. Jiang, V. N. Kravets, and N. Z. Lee, "Engineering change order for combinational and sequential design rectification," in *Design, Auto. & Test in Eur. Conf. (DATE)*, 2020, pp. 726–731.

[38] V. Rao, H. Ondricek, P. Kalla, and F. Enescu, "Algebraic techniques for rectification of finite field circuits," in *Proc. IEEE/IFIP Intl. Conf. on VLSI (VLSI-SoC)*, October 2021, p. To Appear.

[39] ——, "Rectification of integer arithmetic circuits using computer algebra techniques," in *Proc. IEEE Intl. Conf. on Computer Design (ICCD)*, October 2021, p. To Appear.

[40] D. Liu, C. Yu, X. Zhang, and D. Holcomb, "Oracle-guided Incremental sat Solving to Reverse Engineer Camouflaged Logic Circuits," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016.