Next-Generation Automatic Human-Readable Proofs Enabling Polynomial Formal Verification

Rolf Drechsler
University of Bremen
Cyber-Physical Systems, DFKI GmbH
Bremen, Germany
drechsler@uni-bremen.de

Martha Schnieber
University of Bremen
Bremen, Germany
schnieber@uni-bremen.de

ABSTRACT

Within the past years, the complexity of digital circuits has grown significantly, resulting in an increased difficulty of their verification. Only based on formal verification techniques, the correctness of a circuit can be fully guaranteed. However, the verification of circuits using formal verification techniques generally requires exponential time and space in the worst case. During the verification process, the formal representations of functions, e.g. BDDs, can have an exponential size, resulting in an exponential verification complexity.

Thus, recently the concept of *Polynomial Formal Verification* (PFV) has been introduced, where polynomial upper bounds are proven for the verification complexity. This has been done successfully e.g. for adders and multipliers. Polynomial upper bounds have been proven manually, but it has not been researched yet, how they can be proven automatically. Here, we propose a tool that automatically proves polynomial upper bounds, providing human-readable proofs by induction. In this paper, the concept of automatic proofs for PFV is shown using BDDs as an example.

CCS CONCEPTS

• **Hardware → Functional verification**.

KEYWORDS

polynomial formal verification, complexity, automatic proof, binary decision diagrams

ACM Reference Format:
Rolf Drechsler and Martha Schnieber. 2023. Next-Generation Automatic Human-Readable Proofs Enabling Polynomial Formal Verification. In *21st ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '23), September 21–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3610579.3612941

1 INTRODUCTION

With the rising complexity of digital circuits, their verification poses an increasingly difficult challenge. While simulation techniques fail to fully guarantee the correctness of a circuit, formal verification methods, such as techniques based on *Binary Decision Diagrams* (BDDs) [5, 10] or *Boolean Satisfiability* (SAT) [16], can prove the correctness of a circuit. Other formal verification techniques include *Kronecker Functional Decision Diagrams* (KFDDs) [11], *Multiplicative Binary Moment Diagrams* (*BMDs) [7], *Symbolic Computer Algebra* (SCA) [2], or *Answer Set Programming* (ASP) [4]. However, formal verification generally may require exponential time and space in the worst case, as the formal representations, such as decision diagrams, can have an exponential size during the verification process [6]. Due to the unpredictability of the size and therefore the entire verification complexity, recent research has introduced the concept of PFV [8, 13], where polynomial upper bounds are formally proven for the verification time and space complexity.

Several classes of circuits have already been proven to be verifiable in polynomial time and space, e.g. several adder architectures, including the *Ripple Carry Adder* (RCA), *Conditional Sum Adder* (CSA) and *Carry Look Ahead Adder* (CLA) [8, 18], as well as several *Prefix Adders* (PAs) [20]. Additionally, it was proven that BDD circuits, as well as tree-like circuits [9], a simple ALU [14] and symmetric functions [12] can be verified in polynomial time and space using BDDs. Furthermore, PFV using BDDs has been researched for some approximate functions [25] and approximate adders [24]. Other DD types such as KFDDs or *BMDs have also been used for the polynomial formal verification of circuits, e.g. KFDD circuits [23], general tree-like circuits [19] and Wallace-tree like multipliers [15]. Furthermore, SCA [21, 2] and ASP [22] have been applied for PFV as well. However, all previously proven polynomial upper bounds had to be shown manually. This manual task is very time consuming and due to the lack of automation also error-prone.

In this paper, we propose to automate proofs for polynomial upper bounds for the verification process. Automatic proofs have been researched in several fields, e.g. automated theorem proving [17], where logic theorems are automatically proven, or for mathematical problems, such as the four color problem [1], which is solved by automatically checking a finite number of cases. However, automatic proofs checking a finite number of cases are difficult to verify, as no human-readable proof is generated. Thus, our approach yields an automatically generated, but human-readable proof, facilitating the verification of the generated proof. We show the proposed methodology using BDDs. Here, the proofs are conducted with induction.

The paper is structured as follows: In Section 2, the automatic proof engine for BDDs is proposed. In Section 3 based on the presented approach, current limitations and possible extensions are discussed. Finally, the paper is summarized in Section 4.

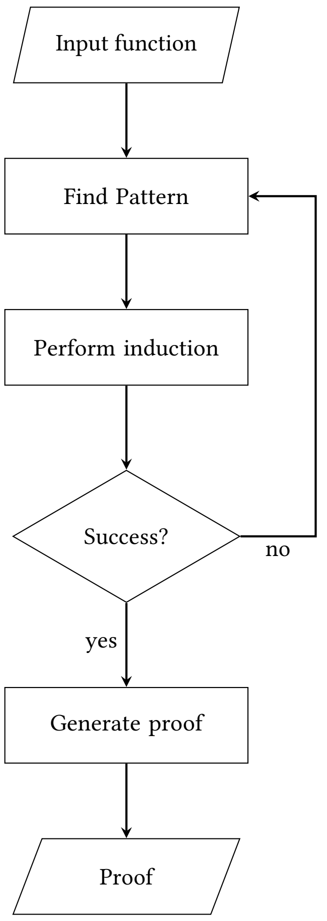

Figure 1: Automatic proof engine flow

2 AUTOMATIC PROOFS

The flow of the proposed tool is shown in Figure 1, where BDDs are used as the underlying formal engine. Firstly, a generalized pattern for the BDD of the input function is guessed by an analysis of the BDD structure, providing the number of added nodes per iteration. Then, a proof by induction is conducted, proving that the generalized pattern fits the input function. If the induction is successful, a human-readable proof is generated. Otherwise, a different pattern is found and the induction proof is repeated.

The tool currently already supports all functions $f : B^n \longrightarrow B$ that can be defined as

$$f(x_1, ..., x_n) = f(x_1, x_{n-i}) \circ g(x_{n-i+1}, ..., x_n),$$

where $\circ \in \{+, \cdot\}$ and $g(x_{n-i+1}, ..., x_n)$ is a function on the input variables $x_{n-i+1}, ..., x_n$ with a constant BDD size and $f(x_1, x_{n-i})$ is not dependent on $x_{n-i+1}, ..., x_n$. We denote the iteration k as the number of applications of $\circ$ performed on a function.

2.1 Generalized Pattern

To prove an upper bound for the BDD size, the general structure of the BDD has to be determined. Here, the pattern specifies the nodes that are added by the k-th iteration. Thus, the generalized pattern

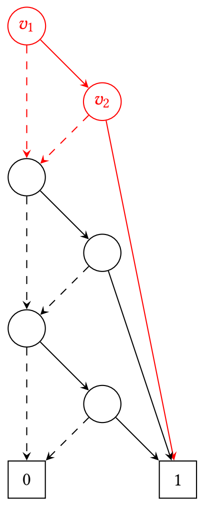

Figure 2: BDD for $f(x_1, ...x_n)$ with $k = 2$

stores the number of nodes added in each iteration, as well as the necessary information for each new node, meaning the nodes to which the new nodes are connected. Furthermore, the BDD for the base case has to be stored in the pattern.

We exemplarily demonstrate the methodology for the function that Bryant introduced in [5]:

$$f(x_1, ...x_n) = \sum_{i=1}^{\frac{n}{2}} x_{2i-1}x_{2i} \tag{1}$$

Firstly, the BDD for the base case x_1x_2 is stored in the pattern. Then, the nodes added by the k-th iteration are determined. For the second iteration $k = 2$, the function is $x_1x_2 + x_3x_4 + x_5x_6$, of which Figure 2 shows the BDD, marking the nodes that are added in the second iteration. Two nodes v_1 and v_2 have to be stored in the pattern, along with the following information:

$$\begin{aligned} low(v_1) &= r(k-1) \\ high(v_1) &= v_2 \\ low(v_2) &= r(k-1) \\ high(v_2) &= 1 \end{aligned}$$

Here, $r(k-1)$ is the root node of the BDD for the $(k-1)$-th iteration. Thus, the first node is the root node, where the *low*-edge leads to the root node of the BDD for $k - 1$, whereas the *high*-edge leads to the second new node. The *low*- and *high*-edge of the second node lead to the root node of the previous iteration and the terminal node 1, respectively. Using the information about the added nodes per iteration, the tool provides a polynomial upper bound for the BDD size. For the k-th iteration of f, this upper bound evaluates to $2k + 4$, as two nodes are added per iteration, whereas the base case consists of 4 nodes.

The proposed tool automatically determines the generalized pattern by an analysis of the BDDs of the function for several values for k. Here, the BDD for k and the BDD for $k - 1$ are compared to determine the new nodes in the BDD for k and their connection to the BDD for $k - 1$.

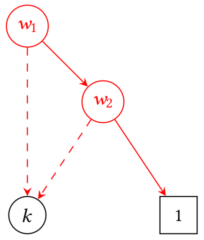

Figure 3: ITE after propagation of ITE operator

2.2 Proof by Induction

If a fitting pattern is found, its correctness is then formally proven by induction on the number of iterations k. For the base case, the BDD is checked and compared to the base case defined by the pattern.

In the induction step, it is proven that the pattern is correct for $k+1$, if it is correct for k. Firstly, the ITE operator [3] is propagated through the function for $k+1$ according to the function description, until all edges lead to nodes from the BDD for k, inserting the induction hypothesis and yielding the BDD for $k+1$, based on the BDD for k. Figure 3 shows the propagation of the ITE operator for $f(x_1, ... x_{2k+2})$, where

$$\begin{aligned} f(x_1, ... x_{2k+2}) &= f(x_1, ... x_{2k}) + x_{2k+1}x_{2k+2} \\ &= ITE(f(x_1, ... x_{2k}), 1, ITE(x_{2k+1}, x_{2k+2})). \end{aligned}$$

Here, two new nodes w_1 and w_2 are added through the propagation.

Afterwards, this BDD is automatically compared to the pattern, proving the correctness or incorrectness of the generalized pattern. Thus, it has to be checked whether each node in the BDD resulting from the propagated ITE operator fits a node from the generalized pattern. For f, w_1 fits v_2, whereas w_2 fits v_2. If the computed BDD matches the generalized pattern, the upper bound given by the pattern is proven.

2.3 Tool Output

After the induction proof, a human-readable proof is generated. As the generalized pattern is intuitive, it is printed in the proof, including the base case and all new nodes per iteration and their respective *high*-edges and *low*-edges. Furthermore, it is displayed as a BDD for the human-readable proof, as shown in Figure 2.

Similarly, the BDD for the base case of the induction proof is displayed. For the induction step, the propagated ITE operator is displayed as a BDD, as shown in Figure 3, followed by a mapping of nodes, where each node from the propagated ITE operator is mapped to a node from the generalized pattern.

3 LIMITATIONS AND CHALLENGES

Currently, the tool implementation covers a limited amount of functions. The implementation of the pattern can only store a constant amount of nodes and therefore, only a constant number of nodes can be added per iteration, leading to BDDs of linear size. Furthermore, the BDD for $k+1$ has to fully include the BDD for k, further limiting the classes of functions for which upper bounds can currently be proven.

The proposed tool can be further expanded to include more functionality, posing numerous challenges and directions for future work.

- The implementation can be expanded to provide automatic proofs for a wider range of more complex functions. The inclusion of more complex functions requires a more complex representation of the generalized pattern.
- The current implementation requires a single output function, for which a proof is conducted. However, the tool could be modified to also provide automatic proofs for a whole class of functions, instead of a single function.
- For BDDs and other DD types, such as KFDDs and BMDs, the variable ordering can significantly influence the DD size, posing a crucial challenge. Currently, the tool operates on a given variable ordering. However, automatically finding a good variable ordering could lower the upper bounds proven by the tool.
- For some functions, e.g. multipliers, BDDs always have an exponential size [6]. Therefore, in future work, the concept can be extended to also include other verification methods. For some circuits, a different verification technique may be more efficient, such as SAT, KFDDs, *BMDs, SCA or ASP. Thus, a future tool could automatically provide an automatic proof for polynomial upper bounds for the verification process of a circuit, automatically finding the best formal verification method for the specific circuit.
- A database of functions for which a proof using any formal verification method could be generated can be established, where the tool guesses a multitude of functions and attempts to provide an automatic proof. If successful with any formal verification method, the function is added to a database.
- If a polynomial upper bound cannot be automatically proven, results from the polynomial formal verification of approximate functions can be applied. If a polynomial upper bound of a different function has been proven and stored in the database, an upper bound of the desired function can be computed according to the number of input assignments for which both functions differ, as has been outlined for BDDs in [25].
- Currently, the presented concept focuses on proving polynomial upper bounds to enable polynomial formal verification. However, for some functions, specific formal verification methods such as BDDs always have an exponential size, e.g. for multipliers [6]. In future work, the tool could also be expanded to be able to prove an exponential lower bound, therefore automatically providing a human-readable proof that PFV using a specific method such as BDDs is not possible for the considered function.

4 CONCLUSION

While all previous results in the topic of *Polynomial Formal Verification* have to be proven manually, in this paper, we have proposed automatically generated human-readable proofs for polynomial upper bounds for PFV. We have shown the concept of automatic proofs on BDDs, where a tool automatically provides a proof for a polynomial upper bound of the BDD size for a function. Furthermore, we have presented the limitations of the current approach, as well as challenges and directions for the future enhancement, e.g. to include a wider class of functions and expand the approach for other formal verification techniques.

ACKNOWLEDGMENTS

Parts of this work have been supported by DFG within the Reinhart Koselleck Project PolyVer: Polynomial Verification of Electronic Circuits (DR 287/36-1).

REFERENCES

[1] Kenneth Appel and Wolfgang Haken. 1976. Every planar map is four colorable. *Bulletin of the American Mathematical Society*, 82, 5.

[2] Mohammed Barhoush, Alireza Mahzoon, and Rolf Drechsler. 2021. Polynomial word-level verification of arithmetic circuits. In *2021 19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 1–9.

[3] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. 1990. Efficient implementation of a BDD package. In *27th ACM/IEEE Design Automation Conference*, 40–45.

[4] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. 2011. Answer set programming at a glance. *Commun. ACM*, 54, 12, 92–103.

[5] Randal E. Bryant. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35, 8, 677–691.

[6] Randal E. Bryant. 1991. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40, 2, 205–213.

[7] Randal E. Bryant and Yirng-An Chen. 1995. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference (DAC)*, 535–541.

[8] Rolf Drechsler. 2021. PolyAdd: polynomial formal verification of adder circuits. In *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 99–104.

[9] Rolf Drechsler. 2021. Polynomial circuit verification using BDDs. In *2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, 49–52.

[10] Rolf Drechsler and Bernd Becker. 2013. *Binary Decision Diagrams: Theory and Implementation*. Springer US.

[11] Rolf Drechsler and Bernd Becker. 1998. Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17, 10, 965–973.

[12] Rolf Drechsler and Caroline Dominik. 2021. Edge verification: ensuring correctness under resource constraints. In *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, 1–6.

[13] Rolf Drechsler and Alireza Mahzoon. 2022. Polynomial formal verification: ensuring correctness under resource constraints : (invited paper). In *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 1–9.

[14] Rolf Drechsler, Alireza Mahzoon, and Lennart Weingarten. 2022. Polynomial formal verification of arithmetic circuits. In *Proceedings of International Conference on Computational Intelligence and Data Engineering*, 457–470.

[15] Martin Keim, Rolf Drechsler, Bernd Becker, Michael Martin, and Paul Molitor. 2003. Polynomial formal verification of multipliers. *Formal Methods in Systen Design*, 22, 1, 39–58.

[16] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. 2002. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21, 12, 1377–1394.

[17] Donald W Loveland. 1978. *Automated Theorem Proving: A Logical Basis (Fundamental Studies in Computer Science)*. Elsevier North-Holland, Inc.

[18] Alireza Mahzoon and Rolf Drechsler. 2021. Late breaking results: polynomial formal verification of fast adders. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 1376–1377.

[19] Alireza Mahzoon and Rolf Drechsler. 2022. Polynomial formal verification of general tree-like circuits. In *2022 China Semiconductor Technology International Conference (CSTIC)*, 1–4.

[20] Alireza Mahzoon and Rolf Drechsler. 2021. Polynomial formal verification of prefix adders. In *2021 IEEE 30th Asian Test Symposium (ATS)*, 85–90.

[21] Alireza Mahzoon, Daniel Große, Christoph Scholl, and Rolf Drechsler. 2020. Towards formal verification of optimized and industrial multipliers. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 544–549.

[22] Mohamed Nadeem, Jan Kleinekathöfer, and Rolf Drechsler. 2023. Polynomial formal verification of adder circuits using answer set programming. In *2023 Reed-Muller Workshop (RM2023)*.

[23] Martha Schnieber and Rolf Drechsler. 2023. Polynomial formal verification of KFDD circuits. In *2023 21th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*.

[24] Martha Schnieber, Saman Froehlich, and Rolf Drechsler. 2022. Polynomial formal verification of approximate adders. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, 761–768.

[25] Martha Schnieber, Saman Froehlich, and Rolf Drechsler. 2022. Polynomial formal verification of approximate functions. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 92–97.