Rolf Drechsler
Daniel Große *Editors*

Recent Findings in Boolean Techniques

Selected Papers from the 14th International Workshop on Boolean Problems

Springer

Recent Findings in Boolean Techniques

Rolf Drechsler • Daniel Große
Editors

Recent Findings in Boolean Techniques

Selected Papers from the 14th International Workshop on Boolean Problems

Springer

Editors
Rolf Drechsler
University of Bremen/DFKI
Bremen, Germany

Daniel Große
Johannes Kepler University of Linz
Linz, Austria

ISBN 978-3-030-68070-1 ISBN 978-3-030-68071-8 (eBook)
https://doi.org/10.1007/978-3-030-68071-8

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Boolean functions are at the core of computer science and the foundation of today's circuits and systems. The *International Workshop on Boolean Problems* (IWSBP) is a bi-annually held and a well-established forum to discuss the recent advances on problems related to Boolean logic and Boolean algebra. In 2020, the 14th edition of the workshop was held virtually from September 24 to September 25 due to the worldwide pandemic. The workshop provided a forum for researchers and engineers from different disciplines to exchange ideas as well as to discuss problems and solutions. The workshop is devoted to both theoretical discoveries and practical applications. This edited book contains a selection of best papers presented at the workshop and one additional paper. The papers in this volume demonstrate new accomplishments in the theory of Boolean problems. Furthermore, several papers illustrate how these results find their way into important practical applications.

The first two chapters in the book are contributions that resulted from the invited keynotes at the workshop. In Chap. 1, Daniela Kaufmann presents *Formal Verification of Integer Multiplier Circuits using Algebraic Reasoning—A Survey*. In Chap. 2, Victor M. van Santen, Florian Klemme, and Hussam Amrouch write about *The Vital Role of Machine Learning in Developing Emerging Technologies*. The following six chapters are extended manuscripts based on the workshop submissions. In Chap. 3, Bernd Steinbach and Christian Posthoff consider *Fast Optimal Synthesis of Symmetric Index Generation Functions*. Felix Weitkämper targets *Axiomatizing Boolean Differentiation* in Chap. 4. In Chap. 5, Radomir S. Stanković, Milena Stanković, Claudio Moraga, and Jaakko Astola investigate bent functions in *Construction of Binary Bent Functions by FFT-Like Permutation Algorithms*. In Chap. 6, Jan Schmidt and Petr Fišer write about *Nonlinear Codes for Test Patterns Compression: The Old School Way*. D. Michael Miller and Gerhard W. Dueck address *Translation Techniques for Reversible Circuit Synthesis with Positive and Negative Controls* in Chap. 7. In Chap. 8, Claudio Moraga focuses on *Hybrid Control of Toffoli and Peres Gates*. Finally, the book is concluded in Chap. 9 by Alireza Mahzoon, Daniel Große, and Rolf Drechsler with *GenMul: Generating Architecturally Complex Multipliers to Challenge Formal Verification Tools*.

We would like to express our thanks to the program committee of the 14th IWSBP as well as to the organizational team, in particular Alireza Mahzoon, Lisa Jungmann, and Kristiane Schmitt. Furthermore, we thank all the authors of contributed chapters who did a great job in submitting their manuscripts of very high quality. A special thanks goes to the keynote speakers of the workshop, Dr. Daniela Kaufmann (Johannes Kepler University Linz, Austria) and Junior Professor Dr. Hussam Amrouch (University of Stuttgart, Germany). Finally, we would like to thank Nandhakumar Sundar, Brian Halm, Zoe Kennedy, and Charles Glaser from Springer. All this would not have been possible without their steady support.

Bremen, Germany Rolf Drechsler

Linz, Austria Daniel Große
December 2020

Contents

Formal Verification of Integer Multiplier Circuits Using Algebraic Reasoning: A Survey 1
Daniela Kaufmann

The Vital Role of Machine Learning in Developing Emerging Technologies 29
Victor M. van Santen, Florian Klemme, and Hussam Amrouch

Fast Optimal Synthesis of Symmetric Index Generation Functions 59
Bernd Steinbach and Christian Posthoff

Axiomatizing Boolean Differentiation 83
Felix Weitkämper

Construction of Binary Bent Functions by FFT-Like Permutation Algorithms 105
Radomir S. Stanković, Milena Stanković, Claudio Moraga, and Jaakko Astola

Nonlinear Codes for Test Patterns Compression: The Old School Way 125
Jan Schmidt and Petr Fišer

Translation Techniques for Reversible Circuit Synthesis with Positive and Negative Controls 143
D. Michael Miller and Gerhard W. Dueck

Hybrid Control of Toffoli and Peres Gates 167
Claudio Moraga

GENMUL: Generating Architecturally Complex Multipliers to Challenge Formal Verification Tools 177
Alireza Mahzoon, Daniel Große, and Rolf Drechsler

Index 193

Formal Verification of Integer Multiplier Circuits Using Algebraic Reasoning: A Survey

Daniela Kaufmann

1 Introduction

Digital circuits carry out logical operations, which make them an important component in computers and digital systems, because they represent models for various digital components and arithmetic operations. The basic function of a digital circuit is to compute binary digital values for the logical function it implements, given binary values at the input. The computation is usually realized by logic gates that represent simple Boolean functions, such as negation (NOT), conjunction (AND), disjunction (OR), or exclusive disjunction (XOR). These logic gates can be combined to build more complex logical operations. A subclass of digital circuits are combinational logic circuits, where the output of the circuit is a function of the present input only, i.e., the output does not depend on previous input values. Combinational logic is used in computer circuits to perform Boolean algebra. For example, the part of an arithmetic logic unit (ALU) in a CPU, which is responsible for mathematical calculations, is constructed using combinational logic. If a circuit implements an arithmetic operation, it is called an *arithmetic circuit*, which can be further refined to determine specific arithmetic operations such as *adder circuits* or *multiplier circuits*.

Since these circuits are such a crucial part of processors, it is extremely important to guarantee their correctness in order to prevent issues like the famous Pentium FDIV bug [49] that was detected in 1994. This bug affected the floating point unit of early Intel Pentium processors. The division algorithm for floating points used a lookup table to calculate the intermediate quotients. Due to a programming error, five entries of the lookup table contained zero instead of +2. Thus the result was

D. Kaufmann (✉)
Johannes Kepler University Linz, Linz, Austria
e-mail: daniela.kaufmann@jku.at

R. Drechsler, D. Große (eds.), *Recent Findings in Boolean Techniques*,
https://doi.org/10.1007/978-3-030-68071-8_1

incorrect and in the worst case the error could affect the fourth significant digit of a decimal number. Even more than 25 years after detecting this bug, automatically proving the correctness of arithmetic circuits, and especially multiplier circuits, is still considered to be a challenge.

Formal verification can be used to prove or disprove the correctness of a given system with respect to a predefined specification. To this end the system is translated into a mathematical model, and automated decision processes are applied to derive the desired correctness property. The different formal verification approaches are distinguished by the mathematical formalism used in the verification process.

Up to now several solving techniques have been developed for multiplier verification. The first technique that was shown to detect the Pentium bug is based on binary decision diagrams [10], more precisely on binary moment diagrams (BMDs) [13] and variants [14], since their size remains linear in the number of input bits of a multiplier. However, this approach requires structural knowledge of the multipliers [11, 13]. It is important to determine the order in which BMDs are built, because it has tremendous influence on the size and thus performance.

A common approach models the problem as a satisfiability (SAT) problem, where the circuit is translated into a formula in conjunctive normal form (CNF). A large set of such encodings was submitted to the SAT Competition 2016 [7]. The results indicated that verifying CNF miters of multipliers needs exponential-sized resolution proofs [8], which implies exponential run-time of CDCL SAT solvers. For simple multiplier architectures, this conjecture is neglected in theory in [5], where it was shown that ring properties do admit polynomial-sized resolution proofs. Recent work shows that pseudo-Boolean solvers can verify the word-level equivalence of simple multiplier architectures that consist only of half- and full-adders [32]. This method is so far not applicable for more complex architectures.

A further approach is based on the usage of theorem provers, such as ACL2 [26]. Theorem provers in combination with SAT are able to certify industrial multipliers [22]. Typically, theorem provers are not fully automated and require domain knowledge. Recently, progress has been made in the theorem prover ACL2 [52], which now allows automated verification of a large set of multiplier architectures. However, the multipliers have to be given as SVL netlists, which rely on the preservation of hierarchical information of the circuits.

Approaches based on bit-level reverse engineering [45, 50] use arithmetic bit-level representations, which are extracted from the gate-level netlists. They are able to verify simple multipliers, but fail to verify non-trivial multipliers. Methods based on term rewriting [53] require domain knowledge and thus are not fully automated.

The currently most effective technique for automated verification of flattened multipliers is based on computer algebra, e.g., [16, 24, 40]. In this method, all gates of the circuit and its specification are represented by polynomials. If the gate polynomials are ordered according to their topological appearance, they generate a Gröbner basis [12]. Hence, the question whether a multiplier circuit is correct can be answered by reducing the specification by the implied Gröbner basis. The multiplier is correct if and only if the reduction returns zero. The main issue of the general algebraic approach is that the size of the intermediate reduction results increases drastically. Thus, several preprocessing techniques and reduction methods have been developed in recent years [16, 24, 40], which attempt to overcome this issue.

Nonetheless, the verification process might not be error-free. Generating and checking proofs independently increases trust in the results of automated reasoning tools. Polynomial proofs can be obtained as a by-product of verifying multiplier circuits [25, 30] and can be checked by independent proof checking tools.

In this chapter, we survey over the current state of the art in verifying integer multipliers using computer algebra. For verification of Galois field multipliers, we refer to [33, 34, 58, 59]. In Sect. 2, we introduce the technique of circuit verification based on algebraic reasoning and present available proof formats. In Sect. 3, we present recent verification tools and discuss their strategies to overcome the issue of monomial blow-up in the intermediate reduction results. We show available benchmark generators in Sect. 4 and conclude with a comprehensive evaluation in Sect. 5.

2 Circuit Verification Using Computer Algebra

In this section, we introduce multiplier circuits and discuss architectural details. We present the algebraic concepts that are needed in the technique of automated circuit verification using computer algebra. Furthermore, we introduce algebraic proof systems that can be used to validate the correctness of the verification results.

2.1 Multiplier Circuits

A digital circuit implements a logical function and computes binary digital values, given binary values at the input. The computation of the function is realized by logic gates, such as NOT, AND, OR, and XOR. The specification of a circuit is the desired relation between its inputs and outputs. A circuit *fulfills a specification* if for all inputs it produces outputs that match this desired relation. The goal of verification is to formally prove that the circuit fulfills its specification.

In this chapter, we consider gate-level integer multipliers with input bits $a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1} \in \{0, 1\}$ and $2n$ output bits $s_0, \dots, s_{2n-1} \in \{0, 1\}$. If the circuit represents multiplication of unsigned integers, the multiplier is correct if and only if for all possible inputs the specification $\mathcal{U}_n = 0$ holds, where:

$$\mathcal{U}_n = -\sum_{i=0}^{2n-1} 2^i s_i + \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) \tag{1}$$

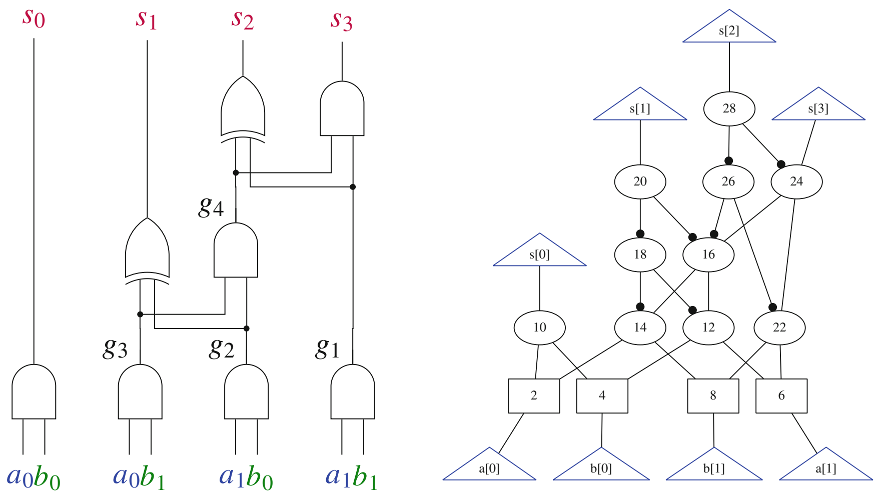

Fig. 1 Gate-level (left) and AIG (right) representation of a 2-bit multiplier circuit [24]

Example 1 The left side of Fig. 1 shows the gate-level representation of a 2-bit unsigned integer multiplier. The variables a_1, a_0, b_1, b_0 represent the input bits of the multiplier and s_3, s_2, s_1, s_0 are the binary outputs of the multiplier. The word-level specification of this circuit is $-8s_3 - 4s_2 - 2s_1 - s_0 + (2b_1 + b_0)(2a_1 + a_0) = 0$.

If the circuit represents signed multiplication, we have to take into account that the integers in the specification $\mathcal{S}_n$ are represented using two's complement.

$$\begin{aligned}\mathcal{S}_n &= -2^{2n-1}s_{2n-1} \\ &\quad + \sum_{i=0}^{2n-2} 2^i s_i - \left(-2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i\right)\left(-2^{n-1}b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i\right)\end{aligned} \tag{2}$$

A common representation of combinational circuits is the encoding as an and-inverter-graph (AIG) [31]. An AIG is a directed acyclic graph, which consists of two-input nodes representing logical conjunction. The edges may contain a marking that indicates logical negation. The AIG representation usually contains more nodes than the gate-level representation but has an unequivocal syntax and semantics and is very efficient to manipulate. The right side of Fig. 1 shows the AIG representation of the gate-level multiplier that is depicted on the left side.

The space and time complexity of a multiplier depends on its architecture. In general, a multiplier circuit can be divided into three parts [44]. In the first component, *partial product generation* (PPG), the partial products a_ib_j for $0 \le i < n$, $0 \le j < n$, as contained in the specification, are generated. This can, for example, be achieved by using simple AND-gates or using a more complex Booth encoding [44].

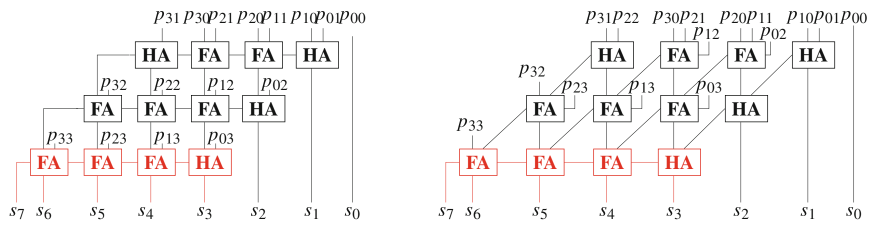
p_{31} p_{30} p_{21} p_{20} p_{11} p_{10} p_{01} p_{00}
HA FA FA HA
p_{32} p_{22} p_{12} p_{02}
FA FA FA HA
p_{33} p_{23} p_{13} p_{03}
FA FA FA HA
s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0
p_{31} p_{22} p_{30} p_{21} p_{12} p_{20} p_{11} p_{02} p_{10} p_{01} p_{00}
HA FA FA HA
p_{32} p_{23} p_{13} p_{03}
FA FA FA HA
p_{33}
FA FA FA HA
s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0

Fig. 2 Architecture of array multipliers (left) and diagonal multipliers (right) [24]

In the second component, *partial product accumulation* (PPA), the partial products are reduced to two layers by multi-operand addition using half-adders (HA), full-adders (FA), and compressors. Well-known accumulation structures are, for example, array or diagonal accumulation, Wallace trees, or compressor trees [44].

In the *final-stage adder* (FSA), the output of the circuit is computed using an adder circuit. Generally, adder circuits can be split into two groups: either the carries are computed alongside the sum bits or they are calculated before the sums. Adders of the first group consist of a sequence of half- and full-adders, giving them a simple but inefficient structure. Examples are ripple-carry or carry-select adders. In order to decrease the latency of carry computation, the adder circuits of the second group precompute the carry bits of the adder. They are called *generate-and-propagate (GP) adders*. Examples are carry look-ahead adders and Kogge-Stone adders [44].

We call multipliers, that can be fully decomposed into half- and full-adders *simple multipliers*, all other architectures are called *complex multipliers*.

Example 2 We show two simple multiplier architectures with input bit-width 4 in Fig. 2. In both circuits, the PPG uses AND-gates, i.e., $p_{ij} = a_i \wedge b_j$. In array multipliers, which are shown on the left side, the partial products are accumulated using a grid-like structure. The multiplier on the right side uses a diagonal structure. In both multipliers, the FSA is a ripple-carry adder, which is highlighted in red.

2.2 Algebra

Let us now briefly summarize algebraic concepts, following [18]. Throughout this section, let $\mathbb{K}[X] = \mathbb{K}[x_1, \ldots, x_n]$ denote the ring of polynomials in variables $x_1, \ldots, x_n$ with coefficients in a field $\mathbb{K}$.

Definition 1 A *term* τ is a product of the form $\tau = x_1^{e_1} \cdots x_n^{e_n}$ for $e_1, \ldots, e_n \in \mathbb{N}$. A *monomial* $m = \alpha\tau$ is a constant multiple of a term, with $\alpha \in \mathbb{K}$. A *polynomial* $p = m_1 + \cdots + m_s$ is a finite sum of monomials.

On the set of terms, an order $\leq$ is fixed such that for all terms τ, σ_1, σ_2 we have $1 \leq \tau$ and $\sigma_1 \leq \sigma_2 \Rightarrow \tau\sigma_1 \leq \tau\sigma_2$. A term order is called a *lexicographic term order* if for all terms $\sigma_1 = x_1^{u_1} \cdots x_n^{u_n}$, $\sigma_2 = x_1^{v_1} \cdots x_n^{v_n}$ we have $\sigma_1 < \sigma_2$ if and only if there exists an index i with $u_j = v_j$ for all $j < i$, and $u_i < v_i$. Every polynomial $p \neq 0$ contains only finitely many terms, the largest of which (with respect to the chosen order $\leq$) is called the *leading term* and denoted by $\text{lt}(p)$. If $p = \alpha\tau + \cdots$ and $\text{lt}(p) = \tau$, then $\text{lc}(p) = \alpha$ is called the *leading coefficient* and $\text{lm}(p) = \alpha\tau$ is called the *leading monomial* of p. We call $p - \alpha\tau$ the *tail* of p.

Definition 2 A nonempty set $I \subseteq \mathbb{K}[X]$ is called an *ideal* if $\forall\ p, q \in I : p + q \in I$ and $\forall\ p \in \mathbb{K}[X]\ \forall\ q \in I : pq \in I$. If $I \subseteq \mathbb{K}[X]$ is an ideal, then a set $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{K}[X]$ is called a *basis* of I if $I = \{q_1 p_1 + \cdots + q_m p_m \mid q_1, \ldots, q_m \in \mathbb{K}[X]\}$. We say *$I$ is generated by P* and write $I = \langle P \rangle$.

The theory of Gröbner bases offers a decision procedure for the so-called ideal membership problem, i.e., given $q \in \mathbb{K}[X]$ and a basis $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{K}[X]$, decide whether q belongs to the ideal generated by $p_1, \ldots, p_m$. If $\{p_1, \ldots, p_m\}$ is a Gröbner basis, then the question can be answered using a multivariate version of polynomial division with remainder (cf. Thm. 3 in Chap. 2 §3 of [18]).

Definition 3 A basis $P = \{p_1, \ldots, p_m\}$ of an ideal $I \subseteq \mathbb{K}[X]$ is called a *Gröbner basis* (with respect to a fixed order $\leq$) if and only if $\forall q \in I \exists p_i \in P : \text{lm}(p_i) \mid \text{lm}(q)$.

Lemma 1 *Every ideal $I \subseteq \mathbb{K}[X]$ has a Gröbner basis with respect to a fixed order $\leq$.*

Proof Cor. 6 in Chap. 2 §5 of [18]. □

Given an arbitrary basis of an ideal, Buchberger's algorithm [12] is able to compute a Gröbner basis for it in finitely many steps.

Lemma 2 *If $P = \{p_1, \ldots, p_m\}$ is a Gröbner basis, then every $f \in \mathbb{K}[X]$ has a unique remainder r with respect to P. Furthermore, it holds that $f - r \in \langle P \rangle$.*

Proof Prop. 1 in Chap. 2 §6 of [18]. □

Ultimately the following Lemma provides the answer on how we can solve the ideal membership problem with the help of Gröbner basis and thus can check whether a polynomial belongs to an ideal or not.

Lemma 3 *Let $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{K}[X]$ be a Gröbner basis, and let $f \in \mathbb{K}[X]$. Then f is contained in the ideal $I = \langle P \rangle$ if and only if the remainder of f with respect to P is zero.*

Proof Cor. 2 in Chap. 2 §6 of [18]. □

2.3 Circuit Verification Using Computer Algebra

In this section we introduce the technique of circuit verification using computer algebra, following [27]. We consider circuits C with inputs $a_0, \dots, a_{n-1}$ and $b_0, \dots, b_{n-1}$, outputs $s_0, \dots, s_{2n-1}$, and a number of logical gates, denoted by $g_1, \dots, g_k$. By R we denote the ring $\mathbb{K}[a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}, g_1, \dots, g_k, s_0, \dots, s_{2n-1}] = \mathbb{K}[X]$.

The semantics of each circuit gate implies a polynomial relation among the input and output variables, such as the following ones:

$$\begin{array}{lll} u = \neg v & \text{implies} & 0 = -u + 1 - v \\ u = v \wedge w & \text{implies} & 0 = -u + vw \\ u = v \vee w & \text{implies} & 0 = -u + v + w - vw \\ u = v \oplus w & \text{implies} & 0 = -u + v + w - 2vw. \end{array} \tag{3}$$

We call these polynomials *gate polynomials* or *gate constraints*. Let $G(C) \subseteq R$ denote the set of polynomials, which contains for each gate of the given circuit the corresponding polynomial of (3).

Example 3 The possible solutions for the gate constraint $p_{00} = a_0 \wedge b_0$ represented as (p_{00}, a_0, b_0) are $(1, 1, 1)$, $(0, 1, 0)$, $(0, 0, 1)$, $(0, 0, 0)$ which are all solutions of the polynomial $-p_{00} + a_0 b_0 = 0$, when a_0, b_0 are restricted to the Boolean domain.

All variables $x \in X$ are Boolean and we enforce this property by assuming the set $B(X) = \{x(1 - x) \mid x \in X\} \subseteq R$ of *Boolean value constraints*.

Since the logical gates are functional, the values of $g_1, \dots, g_k, s_0, \dots, s_{2n-1}$ in a circuit are determined as soon as the inputs $a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1} \in \{0, 1\}$ are fixed. This motivates the following definition of polynomial circuit constraints [27].

Definition 4 Let C be a circuit. A polynomial $p \in R$ is called a *polynomial circuit constraint (PCC)* for C if for every choice of

$$(a_0, \dots, a_{n-1}, b_0, \dots, b_{n-1}) \in \{0, 1\}^{2n}$$

and the resulting values $g_1, \dots, g_k, s_0, \dots, s_{2n-1}$ which are implied by the gates of the circuit C, the substitution of all these values into the polynomial p gives zero. The set of all PCCs for C is denoted by $I(C)$.

It is easy to see that $I(C)$ is an ideal of R. Since it contains all PCCs, this ideal includes all relations that hold among the values at the different points in the circuit. The circuit fulfills a certain specification $\mathcal{L}$ if and only if the polynomial relation corresponding to the specification of the circuit is contained in the ideal $I(C)$.

Thus, checking whether a given circuit C is a correct multiplier reduces to an ideal membership test. Definition 4 does not provide any information of a basis of $I(C)$, hence Gröbner basis technology is not directly applicable. However, we can deduce at least some elements of $I(C)$ from the semantics of circuit gates.

Definition 5 Let C be a circuit and assume $G(C) \subseteq R$ be the set which contains for each gate of C the corresponding polynomial of (3).

Let $B_0(G) = B(\{a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}\})$ and $J(C) = \langle G(C) \cup B_0(G) \rangle \subset R$.

Assume that we have a verifier which checks for a given circuit C and a given specification polynomial $\mathcal{L} \in R$ whether it holds that $\mathcal{L} \in J(C)$. Because it holds that $J(C) = I(C)$ [27], such a verifier is sound and complete.

Theorem 1 *Let C be a circuit, and let $J(C)$ be as in Definition 5. Furthermore, let $\leq$ be a reverse topological lexicographic term order where the variables are ordered such that the variable of a gate output is always greater than the variables attached to the input edges of that gate. Then $G(C) \cup B_0(G)$ is a Gröbner basis for $J(C)$ with respect to the ordering $\leq$.*

Proof This theorem is shown for instance in [27, 34, 54].

Hence $G(C) \cup B_0(G)$ is a Gröbner basis for the ideal $J(C)$ and we can decide membership using Gröbner bases theory, i.e., we reduce the specification $\mathcal{L}$ by elements of $G(C) \cup B_0(G)$ until no further reduction is possible. The circuit is correct if and only if the final remainder is zero.

In this section we restricted the theory to polynomial rings over a field $\mathbb{K}$. A generalization for polynomial rings over *principal ideal domains* (such as $\mathbb{Z}$) can be found in [28], where it is furthermore discussed how to invoke modular reasoning, i.e., reasoning in rings $R = \mathbb{Z}_l[X]$. Modular reasoning allows to eliminate monomials that have large coefficients.

As a final remark, in the case when a polynomial g is not contained in an ideal $I = \langle P \rangle$, i.e., the remainder of dividing g by P is not zero and allows to determine a concrete choice of input assignments for which g does not vanish. In our application of multiplier verification, these evaluations provide counter-examples, in case a circuit is determined not to be a multiplier.

2.4 Algebraic Proof Systems

Although the verification method is sound and complete, it may happen that the implementation contains errors and the reasoning engine delivers wrong results. One way to overcome this issue is to verify the implementation, e.g., [52], which is typically very tedious and requires a lot of effort. Thus, it is common to produce proof certificates in the reasoning engine to monitor the verification process. These proofs are generated as by-product of the reasoning technique and are given to independent (and ideally verified) proof checkers to validate the verification result.

For computer algebra, two algebraic proof systems are used in practice, the *practical algebraic calculus* (PAC) [46], which is based on the *polynomial calculus* (PC) [17], and the *Nullstellensatz proof* format (NSS) [4].

Practical Algebraic Calculus The practical algebraic calculus [46] is an instantiated version of PC [17] which allows efficient proof checking. A PAC proof consists of three components (i) the given set of polynomials G, i.e., the constraint set, (ii) the core proof, i.e., a sequence of proof rules P that model the properties of an ideal, and (iii) the target polynomial f. In a correct proof, it is derived whether the target polynomial can be derived from the constraint set using the proof rules.

Initially the proof format has been defined for polynomial rings $\mathbb{K}[X]$, where $\mathbb{K}$ is a field [17, 46] and all variables represent Boolean values.

The soundness and completeness arguments have been generalized to rings $R[X]$, where all polynomials in the constraint set have unique leading terms that contain only a single variable, cf. Thm. 1 and Thm. 2 in [28]. Recently, the PAC format has been revised to derive a more compact proof representation [30].

Let P be a sequence of polynomials that can be accessed via indices. We write $P(i) = \bot$ to denote that the sequence P at index i does not contain a polynomial, and $P(i \mapsto p)$ to determine that P at index i is set to p. The initial state is $(X = \text{Var}\,(G \cup \{f\}), P)$ where P maps indices to polynomials of G.

[ADD (i, j, k, p)] $\quad (X, P) \implies (X, P(i \mapsto p))$

where $P(j) \neq \bot$, $P(k) \neq \bot$, $P(i) = \bot$, $p \in R[X]/\langle B(X)\rangle$, and $p = P(j) + P(k)$.

[MULT (i, j, q, p)] $\quad (X, P) \implies (X, P(i \mapsto p))$

where $P(j) \neq \bot$, $P(i) = \bot$, $p, q \in R[X]/\langle B(X)\rangle$, and $p = q \cdot P(j)$.

PAC proofs that are defined over $\mathbb{Z}[X]$ can be checked by the checkers PACHECK (implemented in C) [30] and PASTÈQUE (verified in Isabelle/HOL) [30].

Nullstellensatz The Nullstellensatz proof system [4] allows to derive whether a target polynomial $f \in R[X]$ can be represented as a linear combination from a given set of polynomials $G = \{g_1, \ldots, g_l\} \subseteq R[X]$ and the Boolean value constraints $B(X)$. Similar to PAC, the NSS proof system is initially defined for polynomial rings over fields [4]. By the same arguments given for PAC, the soundness and completeness arguments can be generalized for rings $R[X]$ where all polynomials in G have unique leading terms that contain only one variable [25].

Again, we handle the Boolean value constraints implicitly and derive the following proof format. For a polynomial $f \in R[X]/\langle B(X)\rangle$ and a given set of polynomials $G = \{g_1, \ldots, g_l\} \subseteq R[X]/\langle B(X)\rangle$, an NSS proof is an equality P, such that

$$\sum_{i=1}^{l} h_i g_i = f \in R[X]/\langle B(X)\rangle, \tag{4}$$

with $h_i \in R[X]/\langle B(X)\rangle$.

Nullstellensatz proofs over $\mathbb{Z}[X]$ can be checked using NUSS-CHECKER [25].

3 Verification Tools

In this section we present reasoning tools for verification of flattened gate-level integer multipliers using computer algebra. We focus on the most recent work that has been developed in the last 3 years and consider the tools from Yu et al.: ABC/ARTi [16, 60]; Kaufmann et al.: AMulet [28]; and Mahzoon et al.: PolyCleaner [38], RevSCA/RevSCA-2.0 [40], DyPoSub [42] (sorted chronologically).

We discuss the scope of application of these tools and present their techniques that help to overcome the issue of monomial blow-up during reduction. All these tools are considered in the experimental evaluation in Sect. 5.

3.1 Algebraic RewriTing in ABC [15, 16, 57, 60]

The authors of [15, 57] use a method called *function extraction* to verify circuits. Function extraction is a similar algebraic approach to Gröbner basis reduction as presented in Sect. 2. The difference to Gröbner basis reduction is that it is not required to provide the complete specification polynomial of the circuit for reduction. Instead the word-level output of the circuit, i.e., the bit-vector $\sum_{i=0}^{2n-1} 2^i s_i$ for unsigned numbers resp. $-2^{2n-1}s_{2n-1} + \sum_{i=0}^{2n-2} 2^i s_i$ for signed number representation, is reduced by the gate constraints of the given circuit. The Boolean value constraints are reduced implicitly, i.e., every exponent greater than one is immediately reduced to one. This method returns a unique polynomial representation of the functionality of the circuit in terms of the circuit inputs. In order to verify correctness of a circuit, this remainder polynomial needs to be compared to the desired circuit functionality.

In follow-up works [16, 60], the authors introduced an optimization, where half- and full-adders are extracted by identifying subcircuits in the given circuit that represent MAJ3 and XOR3 gates. These XOR3 and MAJ3 gates are essential components of adder trees that are present in most arithmetic circuits. The polynomial constraints of all circuit gates that belong to a MAJ3 or XOR3 gate are replaced by a single polynomial that encodes a MAJ3 or XOR3 gate in order to simplify the polynomial representation of the circuit. These polynomials are sorted topologically pairwise.

The authors developed a framework called ARTi (Algebraic RewriTing) [56] that is integrated within the ABC tool [6]. Verification of multipliers that are given as AIGs is executed using the command `&polyn`, which can be configured to define whether signed or unsigned multiplication is considered. The extraction of the adder trees is invoked by the command `&atree` [16, 60].

This technique is able to handle very large multipliers that can be fully decomposed into half- and full-adders (for instance, the array and diagonal multipliers of Fig. 2) efficiently, but fails on slightly optimized multiplier architectures, because invoking the command `&atree` on these multipliers leads to incompleteness. In the experimental evaluation, we use `&atree` only for those benchmarks where we know that the circuit can be represented by adder trees, i.e., the experiments of Sect. 5.5.

3.2 AMULET *[28]*

In [28], it is presented how the verification approach can be generalized to polynomial rings that include modular reasoning, i.e., $\mathbb{Z}_l[X]$ for $l = 2^{2n}$. This allows to cancel monomials in the intermediate results with coefficients that are multiples of 2^{2n}.

The technique of [28] uses an incremental verification algorithm for circuit verification. In this method, the multiplier circuit is divided into column-wise slices and the specification polynomial is split into multiple polynomials. The correctness of the circuit is shown by incrementally verifying the correctness of each slice. The main advantage of this approach is that only one small part of the global specification is used for reduction, which helps to reduce the size of the intermediate results.

Furthermore, variable elimination is applied before reduction, i.e., after assigning the gates to slices, all variables that occur in only one other polynomial within the same slice are eliminated. Structures that implement a Booth encoding are detected by pattern matching, and their internal gates are eliminated too.

After variable elimination, the column-wise specifications are reduced by the rewritten Gröbner basis until completion. However, certain parts of the multiplier, more precisely particular final stage adders, are hard to verify using computer algebra. These adders usually contain sequences of OR-gates, which lead to an exponential blow-up of the intermediate reduction results. On the other hand, equivalence checking of adders is easy for SAT [29].

We will take a quick excursion and introduce the SAT problem following [19]:

- A *literal* l is either a positive Boolean variable x or its negation $\overline{x}$.
- A *clause* C is a finite disjunction of literals.
- A *formula in conjunctive normal form* (CNF) F is a finite conjunction of clauses.
- An *assignment* τ is a function that consistently maps the literals of F to $v \in \{\mathbf{t}, \mathbf{f}\}$, such that $\tau(x) = v \Leftrightarrow \tau(\overline{x}) = \neg v$, where $\neg\mathbf{t} = \mathbf{f}$ and $\neg\mathbf{f} = \mathbf{t}$.
- A formula evaluates to $\mathbf{t}$ if and only if every clause in the formula evaluates to $\mathbf{t}$. A clause C evaluates to $\mathbf{t}$ if $\exists l \in C$ with $\tau(l) = \mathbf{t}$. Given a CNF formula F, the SAT problem is to decide if there exists an assignment such that F evaluates to $\mathbf{t}$. If such an assignment exists, the formula is *satisfiable*, otherwise it is *unsatisfiable*.

Based on the observation, the technique of [28] combines SAT and computer algebra. It is detected whether a multiplier contains a complex final stage adder, which is then replaced by a simple ripple-carry adder. A bit-level miter, which is expected to be unsatisfiable, is produced to verify the correctness of the replacement using SAT solvers, and the rewritten multiplier is verified by computer algebra techniques.

The authors implemented a tool called AMULET [28] that is able to handle signed and unsigned multipliers given as AIGs. The tool automatically applies adder substitution and verifies the (rewritten) multiplier using computer algebra. Furthermore, AMULET is so far the only tool that is able to produce proof certificates in the PAC and NSS proof formats, cf. Sect. 2.4. In the experiments of this chapter, we will use the maintained version AMULET 1.5 that is currently available on GitHub [23].

3.3 POLYCLEANER *[38]*

The work of [38] provides an extensive analysis why the number of monomials in the reduction results increases drastically, when no preprocessing is applied. The reason of the size explosion are certain monomials, called vanishing monomials, that reduce to zero later in the reduction process. The authors observe that the vanishing monomials origin from gates where the sum and carry output of a half-adder converge and the vanishing monomials remain in the intermediate reduction results until all internal gate polynomials of the half-adders have been substituted.

The work of [38] proposes a method where the converging gates are identified, and the vanishing monomials are locally removed before the specification is reduced. First, for each occurring half-adder in the circuit, possible converging gates are identified and the belonging input cones are determined. A polynomial is extracted for each converging gate by substituting the gate polynomials of the associated cone. Since the extracted polynomial contains the product of the sum and carry output of the corresponding half-adder, it contains the vanishing monomials. After locally removing these vanishing monomials, the specification polynomial is reduced by these vanishing-free polynomials to verify the circuit.

This method is implemented in the tool POLYCLEANER [37] that is able to verify unsigned multipliers that are given as flattened Verilog modules.

3.4 REVSCA/REVSCA-2.0 *[40]*

The paper [40] is a follow-up work on [38]. The authors elaborate on the disadvantages of the proposed method of [38]. First, the method of [38] highly depends on the detection of the half-adders that are considered to be implemented as pairs of XOR and AND gates. The second disadvantage is that the search space for finding the converging gates is very large. Consequently, the method of [38] only works for those multipliers where all half-adders can be detected and fails for more complex multiplier circuits.

In [40] the authors generalize the detection of converging gates and propose a technique that identifies so-called atomic blocks of the multiplier, i.e., half-adders, full-adders, and compressors, using reverse engineering. The detection of converging gates becomes more independent of the actual design of the atomic blocks and the search space to identify these gates can be limited. Furthermore, since not only half-adders are considered as atomic blocks, vanishing-free polynomials are not only generated for converging gate cones, but also for the outputs of atomic blocks and lead to a more compact polynomial representation.

The authors implemented a tool called RevSCA [39] that verifies unsigned multipliers given as AIGs. The implementation has been improved and additionally verification of signed multipliers is supported in RevSCA-2.0 [39].

3.5 DyPoSub *[42]*

In contrast to the already described methods, the technique of [42], a follow-up work of [40], explicitly tries to tackle the problem of verifying multiplier circuits, where logic synthesis and technology mapping are applied.

The problem with these optimized multipliers is that the clear boundaries of certain substructures, such as internal half- and full-adders, may be blurred. Consequently, the compact representation of these internal substructures are no longer available. For example, the discussed methods of Sects. 3.1 and 3.2 heavily rely on these boundaries, either during rewriting or defining the substitution order.

The method described in [42] tries to overcome this issue by using a *dynamic substitution order* that allows to keep the size of the intermediate reduction results on a moderate level. Before reduction is applied, the circuit gates are preprocessed as described in [40], where atomic blocks and converging gate cones are identified and vanishing-free polynomials are extracted for these cones and atomic blocks.

After preprocessing, a dynamic backward rewriting approach is applied to verify the circuit. The core idea is that for each substitution step, a number of candidate polynomials is available, which maintains the overall topological sorting and guarantees that the polynomials of atomic blocks are substituted consecutively. This ensures that the gate polynomials only need to be considered once during reduction.

At each backward rewriting step, there may be several such possible candidates available. The dynamic backward rewriting chooses the reduction candidate by the number of the occurrences of the leading term in the intermediate reduction result in ascending order. After each substitution step, the increase in the number of monomials is checked. If the number of monomials grows by more than 10%, the step is undone and the specification is reduced by the next candidate polynomial in line. If there is no reduction of any candidate satisfying the threshold limit, the threshold limit is increased and the process is repeated from the first candidate.

This approach is implemented in the tool DyPoSub [41] that verifies unsigned multipliers given as AIGs. The experimental data of [42] shows that this technique