Verifying Large Multipliers by Combining SAT and Computer Algebra

Daniela Kaufmann Armin Biere Manuel Kauers
Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
daniela.kaufmann@jku.at armin.biere@jku.at manuel.kauers@jku.at

***Abstract*—We combine SAT and computer algebra to substantially improve the most effective approach for automatically verifying integer multipliers. In our approach complex final stage adders are detected and replaced by simple adders. These simplified multipliers are verified by computer algebra techniques and correctness of the replacement step by SAT solvers. Our new dedicated reduction engine relies on a Gröbner basis theory for coefficient rings which in contrast to previous work no longer are required to be fields. Modular reasoning allows us to verify not only large unsigned and signed multipliers much more efficiently but also truncated multipliers. We are further able to generate and check proofs an order of magnitude faster than in our previous work, relative to verification time, while other competing approaches do not provide certificates.**

I. Introduction

Automated formal verification of arithmetic circuits, most prominently multiplier circuits, remains an important problem, which in practice still requires substantial manual effort. Currently the most effective approach for automatically verifying integer multipliers is based on polynomial reasoning using computer algebra techniques [5], [12], [17], [18], [28].

However, parts of multipliers, i.e., final stage adders, are a real challenge for the computer algebraic approach. In certain adder designs carries are computed by complex tree structures, leading to an explosion of intermediate results. Contrarily SAT solvers can easily verify the equivalence of adders. Therefore we replace complex final stage adders by simpler adders and verify the correctness of the replacement using SAT solvers. The simplified multiplier is verified using computer algebra.

Our new dedicated reduction engine makes use of the structure of the polynomial representation of circuits and is more capable in multiplier verification than computer algebra systems [6], [26] used in our previous work. Additionally it efficiently produces certificates, which validate the correctness of the verification. In previous work we used $\mathbb{Q}$ as coefficient domain. Our experiments show that it is beneficial to use more general rings to support modular arithmetic [25]. We summarize the theory and provide arguments for the correctness of the algebraic approach over more general rings. As a consequence we are able to verify not only unsigned and signed multipliers more efficiently but also truncated multipliers.

Recently significant progress has been made in fully automated verifying integer multipliers using computer algebra. The authors of [17], [18] presented a method allowing local cancellation of vanishing monomials in converging gate cones. This approach is empirically much more successful than previous work in verifying a large variety of multiplier architectures but its formal exposition has room for improvement. The technique of [5], [28] eliminates redundant polynomials by identifying and rewriting half- and full-adders in the circuit. This approach is able to verifying large clean multiplier circuits, but fails on complex multiplier architectures. None of these methods produces certificates. Contrarily theorem provers in combination with SAT are able to certify industrial multipliers [11], however this approach is not fully automated.

II. Specifying Multiplier Circuits

A circuit implements a logical function and the specification of a circuit is a desired relation between the inputs and the outputs of a circuit. We say that 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 and hence deriving correctness.

We consider acyclic gate-level circuits C with inputs $a_0, \ldots, a_{k-1}$, outputs $s_0, \ldots, s_{m-1}$ in $\{0,1\}$, and a number of internal logical gates $g_1, \ldots, g_l \in \{0,1\}$. Thus we fix X to denote the variables $a_0, \ldots, a_{k-1}, g_1, \ldots, g_l, s_0, \ldots, s_{m-1}$. In the algebraic verification approach every input and output of a gate in the circuit is labeled by a variable and for each gate there is a polynomial describing the relation of the input and output variables of the gate. Correctness of the circuit is shown by proving that the specification, encoded as a polynomial $\mathcal{L}$, is implied by the polynomial relations of the gates.

Part of the specification is the ring to which the polynomial $\mathcal{L}$ belongs. The circuit polynomials are also considered as elements of this ring. In our previous work [12] we chose the ring $\mathbb{Q}[X]$. We will now generalize the formulation of circuit verification using computer algebra to other polynomial rings. Our experiments show that by doing so we gain an enormous speed-up in the computation time. Although not formally introducing the theory, related work also relies on more general rings than $\mathbb{Q}[X]$. Furthermore by generalizing the theory to arbitrary polynomial rings we are able to verify different types of multipliers, which we now present. We fix a polynomial ring $R[X]$ and state the corresponding specification of the multipliers as an element of $R[X]$.

Definition 1. Let $\varphi\colon X \to \{0,1\} \subseteq R$ denote an *assignment* of all variables X. We extend φ to an evaluation of polynomials in the natural way, i.e., $\varphi\colon R[X] \to R$.

Supported by Austrian Science Fund (FWF), NFN S11408-N23 (RiSE), P31571-N32, SFB F5004, LIT AI Lab funded by the state of Upper Austria.

To capture multiplication of unsigned integers we consider circuits with two unsigned binary input bitvectors $A = a_{n-1}, \dots, a_0$ and $B = b_{n-1}, \dots, b_0 \in \{0,1\}^n$ and an output bitvector $S = s_{2n-1}, \dots, s_0 \in \{0,1\}^{2n}$, calculating $A \cdot B = S$.

In previous work [22] we verified unsigned integer multipliers, which most naturally are specified over $\mathbb{Z}$.

Definition 2. The word-level specification $\mathcal{U}_n$ of n-bit unsigned integer multipliers in the ring $\mathbb{Z}[X]$ is given as

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

A common way to represent signed integers is using two's complement. The value w of a bitvector $K = k_{n-1}, \dots, k_0$ of length n in two's complement is given as

$$w = -2^{n-1}k_{n-1} + \sum_{i=0}^{n-2} 2^i k_i.$$

Thus in specifying signed multipliers we interpret A, B and S as *signed* bitvectors in two's complement representation.

Definition 3. The specification $\mathcal{S}_n$ of n-bit signed integer multipliers in the ring $\mathbb{Z}[X]$ is given as

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

Our experiments will show that it is beneficial to use modular arithmetic to reduce the size of intermediate verification results. For now we still keep $\mathbb{Z}$ as coefficient domain for the specification of unsigned and signed multipliers. Theorem 3 and 4, in addition with Lemma 3 and Lemma 4, will then allow us to use $\mathbb{Z}_{2^{2n}}[X]$ too, because for any assignment φ the range of the specification does not exceed $\pm 2^{2n}$.

In contrast to the multipliers presented so far a truncated multiplier returns only n output bits for input bitwidth n. In the result of multiplying two integers the n most significant bits are simply discarded. Thus a truncated multiplier calculates $A \cdot B = S \bmod 2^n$. We define the specification of *truncated multipliers* to be an element of the ring $\mathbb{Z}_{2^n}[X]$, because $\mathbb{Z}_{2^n}$ is the ring whose multiplication we wish to describe.

Definition 4. The specification $\mathcal{T}_n$ of n-bit truncated multipliers in the ring $\mathbb{Z}_{2^n}[X]$ is given as

$$\mathcal{T}_n = \sum_{i=0}^{n-1} 2^i s_i - \sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} 2^{i+j} a_i b_j$$

Our theory is not limited to these multiplication instances. A further example would be Galois field multipliers, where the specification is an element of $\mathbb{Z}_2[X]/\langle p \rangle$, for a given irreducible polynomial p [16], [27]. Other possible (though perhaps useless) choices are rings of the form $\mathbb{Z}_m[X_1, \dots, X_k]/I$ for some given $m \in \mathbb{N}$ and some given ideal I.

III. Algebra

In our previous work [12] we outlined the underlying theory of circuit verification using computer algebra for polynomial rings over fields. In this section we generalize the theory to be applicable in more general polynomial rings. To this end let R be a commutative ring with unity and let $R[X]$ with $X = \{x_1, \dots, x_r\}$ be the polynomial ring over R. By $R^\times$ we denote the set of multiplicatively invertible elements of R.

Definition 5. A *term* $\tau = x_1^{d_1} \cdots x_r^{d_r}$ is a product of powers of variables for certain $d_1, \dots, d_r \in \mathbb{N}$. We denote the set of terms by $[X]$. A *monomial* is a multiple of a term $c\tau$, with $c \in R$ and a *polynomial* p 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 it holds that $1 \leq \tau$ and $\sigma_1 \leq \sigma_2 \Rightarrow \tau\sigma_1 \leq \tau\sigma_2$. Such an order is called a *lexicographic term order* if for all terms $\sigma_1 = x_1^{d_1} \cdots x_r^{d_r}$, $\sigma_2 = x_1^{e_1} \cdots x_r^{e_r}$ we have $\sigma_1 < \sigma_2$ iff there exists an index i with $d_j = e_j$ for all $j < i$, and $d_i < e_i$. For a polynomial $p = c\tau + \cdots$ the largest term τ (w.r.t. $\leq$) is called the *leading term* $\operatorname{lt}(p) = \tau$. Furthermore $\operatorname{lc}(p) = c$ is called the *leading coefficient* and $\operatorname{lm}(p) = c\tau$ is called the *leading monomial* of p. Then we call $p - c\tau$ the *tail of* p.

In the circuit the semantics of the logic gates imply polynomial relations among the variables, such as:

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

The polynomial equations are chosen in such a way that the possible solutions with $u, v, w \in \{0,1\}$ of the polynomials are the solutions of the gate constraints and vice versa. As the left side is always zero, we take the freedom to write f instead of $0 = f$. Note that the coefficients $1, -1$ and 2 are elements of the ring R, where 1 represents the unity of R, -1 represents its additive inverse, and $2 = 1 + 1$. On the set of terms we fix a lexicographic term order, called *reverse topological term order*, such that the output variable of a gate is always greater than the variables attached to the input edges of that gate.

By $G(C) \subseteq R[X]$ we denote the set of *circuit polynomials* which contains for each gate of C the corresponding polynomial of (2). To encode that each variable $x \in X$ represents a boolean value, we further have *boolean value constraints* $x(1-x) = 0$. Let $B(Y) = \{y(1-y) \mid y \in Y\} \subseteq R[X]$ for $Y \subseteq X$, be the set of boolean value constraints for Y. By $\mathcal{L}$ we denote the polynomial in $R[X]$ which models the *specified* relation between the input and outputs of the circuit.

Definition 6. A nonempty subset $I \subseteq R[X]$ is called an *ideal* if $\forall\, p, q \in I : p + q \in I$ and $\forall\, p \in R[X]\ \forall\, q \in I : pq \in I$. A set $P = \{p_1, \dots, p_s\} \subseteq R[X]$ is called a *basis* of I if $I = \{p_1 q_1 + \cdots + p_s q_s \mid q_1, \dots, q_s \in R[X]\}$. We say I is generated by P and write $I = \langle P \rangle$. The sum of two ideals I and J is defined as $I + J = \{p + q \mid p \in I, q \in J\}$.

Note, if $I = \langle P \rangle$ and $J = \langle Q \rangle$ are ideals generated by $P, Q \subseteq R[X]$, then $I + J = \langle P \rangle + \langle Q \rangle = \langle P \cup Q \rangle$.

We show that the question whether $\mathcal{L}$ is implied by the gate polynomials of C and the boolean value constraints can be answered by a so-called ideal membership test:

> "Given a polynomial $q \in R[X]$ and a (finite) set of polynomials $P \subseteq R[X]$, decide whether $q \in \langle P \rangle$."

Definition 7. Let $P \subseteq R[X]$. If for a certain term order, all leading terms of P only consist of a single variable with exponent 1 and are unique and further $\mathrm{lc}(p) \in R^\times$ for all $p \in P$, then we say P has *unique monic leading terms* (UMLT). Let $X_0(P) \subseteq X$ be the set of all variables that do not occur as leading terms in P. We further define $B_0(P) = B(X_0(P))$.

Example 1. The set $P = \{-x + 2y, y - z\} \subseteq \mathbb{Z}[x, y, z]$ has UMLT for the lexicographic term order $x > y > z$. Correspondingly $X_0(P) = \{z\}$ and $B_0(P) = \{-z^2 + z\}$.

In the following these $X_0(P)$ will represent inputs of a circuit and accordingly $B_0(P)$ are the boolean value constraints only on its inputs. Note, in our application the leading coefficients of the polynomials in $G(C)$ are only ± 1. However we prefer the more general statement of Def. 7, allowing that Thm. 1 and Thm. 2 also work for more general settings.

Definition 8. Let $P \subseteq R[X]$ be a finite set of polynomials with UMLT. A polynomial $q \in R[X]$ can be *deduced* from P if $q \in \langle P \rangle + \langle B_0(P) \rangle$. In this case we write $P \vdash_R q$.

Definition 9. For a given set $P \subseteq R[X]$, a *model* is an assignment φ such that for all $p \in P$ we have $\varphi(p) = 0$. For a set $P \subseteq R[X]$ and a polynomial $q \in R[X]$, we write $P \models_R q$ if every model for P is also a model for $\{q\}$, i.e., $P \models_R q \iff \forall \varphi : \forall p \in P : \varphi(p) = 0 \Rightarrow \varphi(q) = 0$.

Note, that for the purpose of this paper, these notions of syntactic "deduction" and semantic "models" are restricted to our application where variables take only boolean values.

Theorem 1 (Soundness)**.** Let $P \subseteq R[X]$ be a finite set of polynomials with UMLT and $q \in R[X]$, then

$$P \vdash_R q \quad \Rightarrow \quad P \models_R q.$$

Proof. If $P \vdash_R q$ then $q \in \langle P \rangle + \langle B_0(P) \rangle$ by definition. This means there are $u_1, \ldots, u_m \in R[X]$ and $v_1, \ldots, v_r \in R[X]$ with $q = u_1 p_1 + \cdots + u_m p_m + v_1 b_1 + \cdots + v_r b_r$, where $p_i \in P$ and $b_i = x_i(x_i - 1) \in B_0(P) \subseteq B(X)$ for $i = 1 \ldots r$. Any assignment φ vanishes on $B(X)$, i.e., $\varphi(b_i) = 0$. If φ is also a model of P then $\varphi(p_i) = 0$ too and as a consequence $\varphi(q) = 0$. Therefore $P \models_R q$, as claimed. □

Completeness is not obvious. Consider for instance that $\{2x\} \models_\mathbb{Z} x$ but $x \notin \langle 2x \rangle$ in $\mathbb{Z}[X]$. Requiring P to have UMLT turns out to be essential (which $\{2x\}$ does not have in $\mathbb{Z}[X]$, because $2 \notin \mathbb{Z}^\times$).

Lemma 1. If $P \models_R p$ and $P \models_R q$ then $P \models_R q \pm p$.

Lemma 2. Let $P \subseteq R[X]$ be a finite set of polynomials with UMLT. Then for all $q \in R[X]$ there exists $p \in \langle P \rangle + \langle B_0(P) \rangle$ and $r \in R[X_0(P)]$ with $q = p + r$, such that the monomials in r have only exponents 1.

Proof. Since P has UMLT, we can replace every occurrence of a leading variable of P in q by the corresponding tail. This process has to terminate because the tail of a polynomial contains only smaller variables and the number of variables in P is finite. Thus at some point only variables in $X_0(P)$ are left which do not occur as leading terms. If these variables occur with exponent larger than one we can use $B_0(P)$ to reduce their exponent to 1, which yields r. All reduction steps to obtain r can be captured by adding polynomials $f \cdot g$ with $f \in R[X]$ and $g \in P \cup B_0(P)$. Their sum gives p. □

Example 2. Let $P \subseteq \mathbb{Z}[x, y, z]$ be as in Ex. 1 and assume $q = 2x^2 + xy + z^2 \in \mathbb{Z}[x, y, z]$. Consequently

$$\begin{aligned} p &= (-2x - 5y)(-x + 2y) + (10y + 10z)(y - z) - 11(-z^2 + z) \\ &= 2x^2 + xy + z^2 - 11z \;\in\; \langle P \rangle + \langle B_0(P) \rangle \;\text{ and} \end{aligned}$$

$$r = 11z \;\in\; \mathbb{Z}[X_0(P)].$$

Theorem 2 (Completeness)**.** Let $P \subseteq R[X]$ be a finite set of polynomials with UMLT. Then for every $q \in R[X]$ we have

$$P \models_R q \quad \Rightarrow \quad P \vdash_R q.$$

Proof. Suppose we have $P \models_R q$. Then our goal is to show $q \in \langle P \rangle + \langle B_0(P)) \rangle$. First, by applying Lemma 2, we obtain $p \in \langle P \rangle + \langle B_0(P) \rangle$ and $r \in R[X_0(P)]$ with $q = p + r$. Thus $P \vdash_R p$ by definition. Using Thm. 1 we derive $P \models_R p$ and accordingly $P \models_R q - p = r$ by Lemma 1. Now assume $r \neq 0$ and let m be a monomial of r which contains the smallest number of variables. Consider the assignment φ that maps $x \in X_0(P)$ to 1 if it appears in m and to 0 otherwise. Therefore $\varphi(r) \neq 0$ since exponents of variables in r are all one. This assignment on $X_0(P)$ admits a unique extension to X which vanishes on P (e.g., if $-x + t \in P$ with leading monomial $-x$, then choose $\varphi(x) = \varphi(t)$). This contradicts $P \models_R r$. Thus $r = 0$ and $q = p + r \in \langle P \rangle + \langle B_0(P)) \rangle$. □

It is easy to see that for an acyclic circuit C the set $G(C)$ has UMLT for the fixed reverse topological term order. As a consequence Theorem 1 and 2 can be applied and show that deciding the correctness of circuits can been reduced to deciding ideal membership problems for $R[X]$.

Definition 10. $\quad I(C) = \{f \in R[X] : G(C) \models_R f\}$.

It also easily follows that $I(C)$ is an ideal and contains all the relations that hold among the values at the different signals (gates and inputs) of the circuit. Thus we are particularly interested whether the specification polynomial $\mathcal{L}$ is in $I(C)$.

Definition 11. A circuit C *fulfills* $\mathcal{L}$ iff $\mathcal{L} \in I(C)$.

Definition 12. Write $B_0(C) = B_0(G(C))$ for an acyclic circuit C and define $J(C) = \langle G(C) \cup B_0(C) \rangle$ in $R[X]$.

Note that $J(C)$ is generated by the gate polynomials $G(C)$ and the boolean value constraints on the variables $X_0(G(C))$ not occurring as leading term in $G(C)$. Further, by definition, $q \in J(C)$ iff $G(C) \vdash_R q$. Thus $J(C)$ contains exactly those polynomial constraints "deducible" from the circuit.

Corollary 1. For all acyclic circuits C, it holds $I(C) = J(C)$.

Proof. By the choice of term order, $G(C)$ satisfies the necessary conditions of Thm. 1 and Thm. 2 and applying them allows to conclude $q \in I(C) \Leftrightarrow q \in J(C)$. □

Corollary 2. A circuit C fulfills $\mathcal{L}$ iff $\mathcal{L} \in J(C)$.

In order to improve efficiency through modular reasoning (replacing $\mathbb{Z}$ by $\mathbb{Z}_{2^{2n}}$) we show that the specifications for unsigned and signed multipliers remain correct for $\mathbb{Z}_{2^{2n}}$ too.

Lemma 3. For all assignments $\varphi\colon X \to \{0,1\}$ it holds that $\varphi(\mathcal{U}_n) \in [-2^{2n}+1, 2^{2n}-1]$ in $\mathbb{Z}$.

Proof. The maximum of $\varphi(\mathcal{U}_n)$ is reached for the assignment $\varphi_{\max}$ with $\varphi_{\max}(s) = 1$ for all $s \in S$ and $\varphi_{\max}(x) = 0$ for $x \in A \cup B$. Consequently

$$\varphi_{\max}(\mathcal{U}_n) = \sum_{i=0}^{2n-1} 2^i = 2^{2n} - 1 < 2^{2n}.$$

The minimum of $\varphi(\mathcal{U}_n)$ is reached for the assignment $\varphi_{\min}$ with $\varphi_{\min}(s) = 0$ for all $s \in S$ and $\varphi_{\min}(x) = 1$ for $x \in A \cup B$. It follows (assuming of course $n > 0$) that

$$\varphi_{\min}(\mathcal{U}_n) = -(2^n-1)^2 = -2^{2n} + \underbrace{2^{n+1}}_{>2} - 1 > -2^{2n}. \qquad \square$$

Lemma 4. For all assignments $\varphi\colon X \to \{0,1\}$ it holds that $\varphi(\mathcal{S}_n) \in [-2^{2n}+1, 2^{2n}-1]$ in $\mathbb{Z}$.

Proof. The maximum of $\varphi(\mathcal{S}_n)$ is reached for the assignment $\varphi_{\max}$ with $\varphi_{\max}(s_i) = 1$ for all $0 \le i \le 2n-2$ and $\varphi_{\max}(s_{2n-1}) = 0$ and $\varphi_{\max}(a_j) = 1$ for all $0 \le j \le n-2$ and $\varphi_{\max}(a_{n-1}) = 0$ and $\varphi_{\max}(b_j) = 0$ for all $0 \le j \le n-2$ and $\varphi_{\max}(b_{n-1}) = 1$. Then

$$\varphi_{\max}(\mathcal{S}_n) = 2^{2n-1} - 1 + 2^{n-1}(2^{n-1} - 1).$$

By transforming the inequality we gain the desired result.

$$2^{2n-1} + 2^{2n-2} - 2^{n-1} - 1 < 2^{2n}$$
$$2^{2n-2}(2 + 1 - 4) - 2^{n-1} - 1 < 0$$

The minimum of $\varphi(\mathcal{S}_n)$ is reached for the assignment $\varphi_{\min}$ with $\varphi_{\min}(s_i) = 0$ for all $0 \le i \le 2n-2$ and $\varphi_{\min}(s_{2n-1}) = 1$ and $\varphi_{\min}(a_j) = \varphi_{\min}(b_j) = 0$ for all $0 \le j \le n-2$ and $\varphi_{\min}(a_{n-1}) = \varphi_{\min}(b_{n-1}) = 1$. It follows that

$$\begin{aligned}\varphi_{\min}(\mathcal{S}_n) &= -2^{2n-1} - (-2^{n-1})^2 \\ &= -3 \cdot 2^{2n-2} > -4 \cdot 2^{2n-2} = -2^{2n}. \qquad \square\end{aligned}$$

Theorem 3. $G(C) \models_{\mathbb{Z}} \mathcal{U}_n$ iff $G(C) \models_{\mathbb{Z}_{2^{2n}}} \mathcal{U}_n$.

Theorem 4. $G(C) \models_{\mathbb{Z}} \mathcal{S}_n$ iff $G(C) \models_{\mathbb{Z}_{2^{2n}}} \mathcal{S}_n$.

IV. D-Gröbner bases

The question whether a circuit fulfills a given specification can be answered by an ideal membership test. The theory of Gröbner bases [4] offers a decision procedure for this problem. For the polynomial rings applied in Sect II, we use the more general theory of D-Gröbner bases [2], where the coefficient ring is a *principal ideal domain* (PID). Let D be a PID.

Some facts about the theory of D-Gröbner bases are:

- Let $p, q, r \in D[X]$. We say q *D-reduces* to r w.r.t. p if there exists a monomial m' in q with $m' = m\,\mathrm{lm}(p)$ and $r = q - mp$. If $m' = \mathrm{lm}(q)$, we call this *top-D-reduction*.
- Let $q \in D[X]$ and $P \subseteq D[X]$. The remainder r of the D-reduction of q by P is such that $q - r \in \langle P \rangle$ and r is *D-reduced* w.r.t. P. If r is calculated using only top-D-reductions, then r is *top-D-reduced* w.r.t. P.
- Let $q \in D[X]$ and $P \subseteq D[X]$ with UMLT. If $lc(p) = \pm 1$ for $p \in P$, then D-reduction of q w.r.t. p amounts to replacing every occurrence of $lt(p)$ in q by the tail of p.
- A basis P of an ideal $I \subseteq D[X]$ is called a *D-Gröbner basis* of I iff $\forall q \in I\ \exists p \in P : \mathrm{lm}(p) \mid \mathrm{lm}(q)$.
- Every ideal of $D[X]$ has a D-Gröbner basis, and there is an algorithm (Thm 10.14 of [2]) which, given an arbitrary basis of an ideal, computes a D-Gröbner basis of it in finitely many steps. It is based on repeated computation of so-called *S-polynomials* and *G-polynomials*.

Definition 13. Let $g_1, g_2 \in D[X]$. Assume

$\mathrm{lcm}(\mathrm{lc}(g_1), \mathrm{lc}(g_2)) = b_1 \mathrm{lc}(g_1) = b_2 \mathrm{lc}(g_2)$ with $b_i \in D$ and
$\mathrm{lcm}(\mathrm{lt}(g_1), \mathrm{lt}(g_2)) = s_1 \mathrm{lt}(g_1) = s_2 \mathrm{lt}(g_2)$ with $s_i \in [X]$ and

lcm the least common multiple. Further pick $c_1, c_2 \in D$ such that $c = \gcd(\mathrm{lc}(g_1), \mathrm{lc}(g_2)) = c_1\,\mathrm{lc}(g_1) + c_2\,\mathrm{lc}(g_2)$, with gcd the greatest common divisor. Then define

$$\begin{aligned}\mathrm{spol}(g_1, g_2) &:= b_1 s_1 g_1 - b_2 s_2 g_2 \\ \mathrm{gpol}(g_1, g_2) &:= c_1 s_1 g_1 + c_2 s_2 g_2\end{aligned}$$

Lemma 5. [Cor. 10.12 in [2]] A set $P \subseteq D[X]$ is a D-Gröbner basis of $\langle P \rangle$ iff for all pairs $(p_1, p_2) \in P \times P$ the remainder of D-reducing $\mathrm{spol}(p_1, p_2)$ w.r.t. P is zero and $\mathrm{gpol}(p_1, p_2)$ top-D-reduces to zero w.r.t. P.

Lemma 6. [Thm.11 in [15]] Let $p_1, p_2 \in D[X]$ be such that $\mathrm{lcm}(\mathrm{lt}(p_1), \mathrm{lt}(p_2)) = \mathrm{lt}(p_1)\,\mathrm{lt}(p_2)$. If $\mathrm{lc}(p_1) \mid \mathrm{lc}(p_2)$ then $\mathrm{spol}(p_1, p_2)$ and $\mathrm{gpol}(p_1, p_2)$ (top-)D-reduce to zero.

A. D-Gröbner bases applied to Multiplier Verification

We use Lemma 6 to derive a D-Gröbner basis of the ideal $J(C)$. The following theorem shows that we neither have to compute S-polynomials nor G-polynomials.

Theorem 5. Let R be a PID and $J(C) = \langle G(C) \cup B_0(C) \rangle$ be as in Def. 12. Then $G(C) \cup B_0(C)$ is a D-Gröbner basis of $J(C)$ for $R = D$.

Proof. Since $G(C)$ has UMLT, $G(C) \cup B_0(C)$ has UMLT and thus it holds by Lemma 6, (top-)D-reduction of $\mathrm{spol}(p, q)$ and $\mathrm{gpol}(p, q)$ by $\{p, q\}$ gives the remainder zero for any choice $p, q \in G(C) \cup B_0(C)$ and by Lemma 5 the claim follows. □

For the multiplier circuits described in Sect II we chose the polynomial rings $\mathbb{Z}_l[X]$ with $l \in \mathbb{N}$. For example for the truncated multiplier we set $l = 2^n$. Unless l is a prime, the ring $\mathbb{Z}_l$ has zero divisors and is therefore not a PID. However the ideal membership test in the ring $\mathbb{Z}_l[X]$ can be reduced to an ideal membership test in the ring $\mathbb{Z}[X]$, and $\mathbb{Z}$ is a PID.

Lemma 7. Let $l \in \mathbb{N}$ and let $I \subseteq \mathbb{Z}[X]$ be an ideal. There is a bijective correspondence from $q \in I + \langle l \rangle \subseteq \mathbb{Z}[X]$ to $[q] \in \{[p] \mid p \in I\} \subseteq \mathbb{Z}[X]/\langle l \rangle$, where $[q]$ is the equivalence class of q. Furthermore $\mathbb{Z}[X]/\langle l \rangle \cong \mathbb{Z}_l[X]$.

Proof. The first claim follows from Prop. 4.3.a Chap. 10 of [1], with $\pi \colon q \mapsto [q]$. The second claim follows from the fundamental theorem of homomorphisms. □

Lemma 7 says that whenever we want to decide whether a polynomial $q \in I \subseteq \mathbb{Z}_l[X]$ we can instead check whether $q \in I + \langle l \rangle \subseteq \mathbb{Z}[X]$. And for the latter we have the concept of D-Gröbner bases available.

Lemma 8. Let C be an acyclic circuit, $l \in \mathbb{N}$. Then $G(C) \cup B_0(C) \cup \{l\}$ is a D-Gröbner basis for $I(C) + \langle l \rangle \subseteq \mathbb{Z}[X]$.

Proof. It remains to show that for all $p \in G(C) \cup B_0(C)$ it holds that $\mathrm{spol}(p, l)$ D-reduces to zero and $\mathrm{gpol}(p, l)$ top-D-reduces to zero, which follows from Lemma 6, because $\mathrm{lc}(p) = -1$ and $\mathrm{lt}(l) = 1$. □

V. Variable elimination

D-Gröbner bases can be used as a kind of black-box for deciding the ideal membership of the circuit specification. However it was shown in [17], [12] that by simply reducing the specification by the polynomials $G(C) \cup B_0(C)$, the size of the intermediate reduction results increases drastically.

In [12] we presented a theorem which allows to simplify local parts of Gröbner bases over fields without changing the rest of the circuit. We extracted specific patterns of the original multiplier, eliminated the internal variables and only the corresponding specification of the sub-circuit remained. In this work we present a more general elimination procedure which subsumes our proposed rewriting methods of [12]. We introduce a technical theorem in the fashion of Thm. 4 of [12], which is applicable in more general polynomial rings.

Definition 14. Let $I \subseteq D[X] = D[Y, z]$ be an ideal. The ideal $I \cap D[Y]$ of $D[Y]$ is called an *elimination ideal* of I.

In general computing a D-Gröbner basis for the elimination ideal means that we explicitly need to compute a D-Gröbner basis w.r.t. a different term order. However if the D-Gröbner basis of I has UMLT, we will show that we can instantly obtain a D-Gröbner basis for the elimination ideal.

Definition 15. Let $P \subseteq D[X]$ be a D-Gröbner basis of $\langle P \rangle$ with UMLT. We say P is *reduced for z* if the variable $z \in X$ is contained in exactly one polynomial $p \in P$ and $\mathrm{lt}(p) = z$.

Lemma 9. Let P be a D-Gröbner basis with UMLT and let $p \in P$. Let $H = P$ be such that all polynomials $h \neq p \in H$ are D-reduced w.r.t. p. Then H is reduced for $\mathrm{lt}(p) = z$.

Proof. Let $f \neq p \in P$ be an arbitrary polynomial containing z. By definition $\mathrm{lt}(f) \neq z$. Let r be the remainder of D-reducing f w.r.t. p. Because $\mathrm{lc}(p) \in D^\times$, r is free of z.

Let $H = (P \setminus \{f\}) \cup \{r\}$. Since $\mathrm{lm}(r) = \mathrm{lm}(f)$ it follows that H is a D-Gröbner basis with UMLT. After repeating the steps for all $f \in P$, p is the only polynomial containing z. □

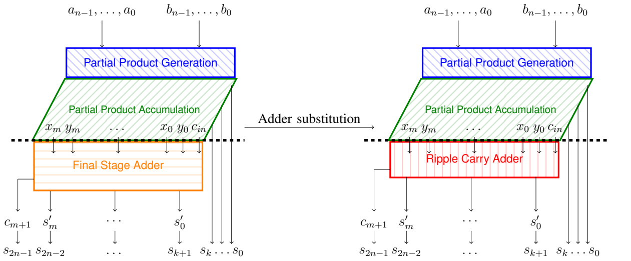
$a_{n-1}, \ldots, a_0$
$b_{n-1}, \ldots, b_0$
Partial Product Generation
Partial Product Accumulation
x_m y_m $\ldots$ x_0 y_0 c_{in}
Final Stage Adder
c_{m+1} s'_m $\ldots$ s'_0
s_{2n-1} s_{2n-2} $\ldots$ s_{k+1} $s_k \ldots s_0$
Adder substitution
$a_{n-1}, \ldots, a_0$
$b_{n-1}, \ldots, b_0$
Partial Product Generation
Partial Product Accumulation
x_m y_m $\ldots$ x_0 y_0 c_{in}
Ripple Carry Adder
c_{m+1} s'_m $\ldots$ s'_0
s_{2n-1} s_{2n-2} $\ldots$ s_{k+1} $s_k \ldots s_0$

Fig. 1. Substituting the final stage adder to gain a simplified multiplier.

Theorem 6. Let $I \subseteq D[X]$ be an ideal. Let P be a D-Gröbner basis of I with UMLT which is reduced for z. Let $p \in P$ be the polynomial with $\mathrm{lt}(p) = z$. Then $P \setminus \{p\}$ is a D-Gröbner basis with UMLT for the ideal $J = I \cap D[X \setminus \{z\}]$.

Proof. We show that $\forall f \in J \ \exists q \in P \setminus \{p\} : \mathrm{lm}(q) \mid \mathrm{lm}(f)$. Since P is a D-Gröbner basis there exists a polynomial $h \in P$ such that $\mathrm{lm}(h) \mid \mathrm{lm}(f)$. Because J is free of z, it follows that $\mathrm{lm}(p) \nmid \mathrm{lm}(f)$. Thus $h \neq p$ and consequently $h \in P \setminus \{p\}$. □

In our approach we apply variable elimination for circuits C as follows. Let $Y = X \setminus \{z\}$. We successively select variables $z \in X$ for elimination and rewrite $G(C) \cup B_0(C)$ according to Lemma 9 such that $G(C) \cup B_0(C)$ is reduced for z. By Thm. 6 removing the polynomial p with $\mathrm{lt}(p) = z$ yields a D-Gröbner basis $(G(C) \setminus \{p\}) \cup B_0(C)$ for $I(C) \cap D[Y]$.

We derive by the proof of Lemma 8 that $(G(C) \setminus \{p\}) \cup B_0(C) \cup \{l\}$ is a D-Gröbner basis for $(I(C) + \langle l \rangle) \cap \mathbb{Z}[Y]$.

VI. Combining SAT and Computer Algebra

In this section we present how to combine the algebraic verification approach with SAT to successfully verify complex multiplier circuits given as And-Inverter-Graphs (AIG) [14].

Multipliers can be decomposed into three components [21], cf. Fig. 1. In the first component *partial product generation* (PPG) the partial products $a_i b_j$ (as contained in $\mathcal{L}$) are generated. This can for example be achieved using quadratically many AND-gates or using a more complex Booth encoding.

The second component *partial product accumulation* (PPA) reduces the partial products to two layers, where full- and half-adders are arranged in different patterns to sum up the partial products. Well known accumulation structures are array accumulation, Wallace trees or compressor trees.

In the *final stage adder* (FSA) the output of the circuit is computed. Generally adder circuits can be divided into two groups, either the carries are computed alongside the sum bits or they are calculated before the sum. Adders of the first group are usually based on a sequence of half- and full-adders, which gives them a simple but inefficient structure. Examples are ripple-carry adders 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 also called generate-and-propagate (GP) adders and examples are carry-lookahead adders and Kogge-Stone adders.

Adders of the second group are hard to verify using the algebraic approach, due to the OR trees needed to precompute carries. Due to the polynomial representation of OR-gates,

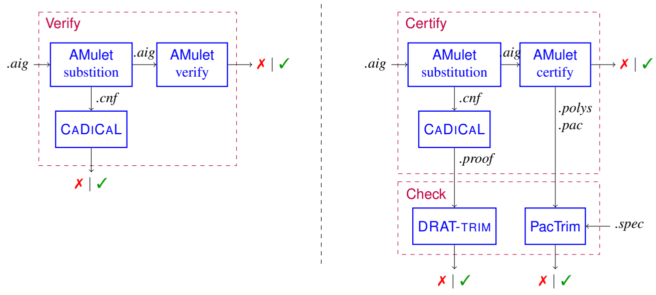

Fig. 2. Tool chain for verification (left) or certification and checking (right).

cf. (2), this leads to an exponential blow-up in the polynomial reduction. During preparation for the SAT Race 2019 [13] we observed that checking the equivalence of different adder circuits is rather trivial for SAT solvers. We use this observation in the verification procedure and determine whether the final stage adder is a GP adder. Figure 2 shows our tool chain used for verifying (left side) and certifying (right side) multiplier circuits given as an And-Inverter-Graph (AIG) [14]. We implemented a new dedicated reduction engine AMulet in C. Adder substitution is automatically applied and, if necessary, a simplified AIG and miter encoded as propositional formula in conjunctive normal form (CNF) are returned.

```
Algorithm 1: Identifying GP adders in AMulet
   Input  : Circuit $C$ in AIG format
   Output: Determine whether $C$ might contain a GP adder
 1 $j \leftarrow 2n-2$, $\tau \leftarrow 1$;
 2 while $\tau$ and $j \geq 0$ do
 3     $\tau, c_j, p_j \leftarrow$ Check-if-XOR-and-Identify-p$_j$-and-c$_j$ ($s_j$);
 4     $x_j, y_j \leftarrow$Declare-Adder-Inputs ($p_j, \tau$);
 5     $j \leftarrow j-1$;
 6 end
 7 $c_{in} \leftarrow c_j$;
 8 for $i \leftarrow j$ to $2n-1$ do
 9     $m \leftarrow$ Follow-and-Mark-Paths($s_i$);
10 end
11 return $m = 0$
```

Detecting GP adders in non-synthesized multipliers is a simple task and the pseudo-code is listed in Alg. 1. In a GP adder with inputs $x_0, \ldots, x_m, y_0, \ldots, y_m$ and outputs $s'_0, \ldots, s'_m, c_{m+1}$, cf. Fig 1, the outputs s'_i are calculated as $s'_i = p_i \oplus c_i$, with $p_i = x_i \oplus y_i$. The carries c_i are recursively generated as $c_i = (x_{i-1} \wedge y_{i-1}) \vee (c_{i-1} \wedge p_{i-1})$. The precise computation of the carries c_i (recursively, unrolled or mixed) depends on the circuit architecture.

If a multiplier contains a GP adder, the most significant output bit s_{2n-1} is the carry output of the adder, i.e. $s_{2n-1} = c_{m+1}$ in Fig. 1. Thus the loop in line 2 of Alg. 1 starts at $2n-2$. At first we check whether the output bit s_i is an XOR-gate, which can easily be identified in AIGs. If s_i is an XOR gate, its inputs are p_i and c_i. We can clearly identify which is which, because p_i has to be an XOR gate, whereas c_i cannot be an XOR gate. In the next step of the loop (line 4) we mark the inputs of the XOR gate p_i as adder inputs x_i and y_i.

If s_i is not an XOR gate or we cannot clearly identify p_i and c_i we stop the loop (indicated by τ), because then s_i is not computed by the GP adder. As can be seen in Fig. 1, some smaller output bits are directly computed in the PPA step. We mark the smallest c_i as the carry-in c_{in} of the GP adder.

In the next phase of our algorithm we follow all input paths of s_i for $j \leq i \leq 2n-1$. We now include s_{2n-1}, because it is the carry-out of the adder. We mark the gates alongside the paths and stop whenever we reach a marked input x_i or y_i or c_{in}. If we encounter a path, which ends at the primary inputs a_i, b_i of the multiplier, then we do not consider the final stage adder as a GP adder.

If we detect a final stage GP adder, we substitute it by a simple ripple-carry adder, which has the same inputs x_i, y_i and c_{in}. We do not change the first two stages of a multiplier, as depicted in Fig. 1. To prove that the ripple-carry adder is equivalent to the GP adder we generate a bitlevel miter in conjunctive normal form, which is verified by a SAT solver (CaDiCaL [3]). If the final stage adder is not a GP adder we do not apply adder substitution.

```
Algorithm 2: Outline of verification flow in AMulet
   Input  : Substituted circuit $C$ in AIG format
   Output: Determine whether $C$ is a multiplier
 1 for $i \leftarrow 0$ to $2n-1$ do
 2     $S_i \leftarrow$ Define-Cone-of-Influence(i);
 3     Order ($S_i$);
 4     Search-for-Booth-Encoding ($S_i$);
 5     Local-Elimination ($S_i$);
 6 end
 7 Global-Elimination ();
 8 $C_0 \leftarrow$ Incremental-Reduction ();
 9 return $C_0 = 0$
```

After substitution we verify or certify the rewritten AIG in AMulet. The outline of the flow is depicted in Alg. 2.

For verification we use our incremental column-wise verification algorithm of [12]. The goal is to split the verification approach into smaller more manageable sub-problems by partitioning the circuit into column-wise slices and by splitting the word-level specification of a multiplier into multiple polynomials which relate the partial products, incoming carries, the sum output bit and the outgoing carries of each slice. The incremental specification presented in [12] is tailored to unsigned multipliers, but it can easily be adapted to more general multiplier specifications by adding coefficients.

We first define slices based on the input cones of the outputs and order the variables in the slices according to their level seen from the circuit inputs (line 2 in Alg. 2). This ensures that the variables are topologically sorted and the corresponding polynomials have UMLT and thus form a D-Gröbner basis.

After sorting we apply syntactic pattern matching to detect whether the circuit uses Booth encoding. In Booth encoding consecutive primary multiplier inputs are used as inputs of XOR-gates which are then combined to form an OR-gate. These XOR- and OR-gates are input to several gates in multiple slices and to increase cancellation of common monomials, we identify corresponding variables.

For local variable elimination (line 4) we loop over the gate polynomials in each slice and eliminate the variables of the leading terms which only occur in polynomials in the same slice and which are contained in exactly one other polynomial inside the slice. We repeatedly apply variable elimination until all variables of leading terms are either contained in the tails of multiple polynomials or occur in polynomials of bigger slices.

After reducing the number of variables inside the slices we eliminate variables which we marked in line 3 of Alg. 2. The difference to local variable elimination is that we now have to consider all polynomials from the circuit.

After variable elimination we apply Alg. 2 of [12] and reduce the column-wise specification by the rewritten sliced D-Gröbner bases and report whether the final result is zero or not. Our tool AMulet uses the UMLT property of the D-Gröbner basis for D-reduction, making it much more efficient than the computer algebra systems [26], [6] used in our previous work, which are designed for more general sets of polynomials. We use the property that every leading monomial contains at most one variable with exponent 1 and with coefficient -1 and thus D-reduction reduces to replacing every occurrence of the leading variable by the tail of the polynomial. As a further optimization we employ reduction by the boolean value constraints implicitly. Whenever a term in the intermediate reduction results contains an exponent larger than 1, we immediately eliminate the exponent, without applying explicit reduction by the corresponding boolean value constraint.

If we want to certify verification we generate PAC proofs [23] in AMulet as by-product of the verification algorithm. These proofs can be checked by our independent proof checker PacTrim [23], cf. right side of Fig. 2. We write proofs as sequences, where each rule is of the following form:

$$\begin{array}{ll} + : p_i, p_j, p_i + p_j; & \begin{array}{l} p_i, p_j \text{ appearing earlier in the proof} \\ \text{or are contained in } G(C) \\ \text{and } p_i + p_j \text{ being reduced by } B(X) \end{array} \\ \\ * : p_i, q, qp_i; & \begin{array}{l} p_i \text{ appearing earlier in the proof} \\ \text{or is contained in } G(C) \\ \text{and } q \in R[X] \text{ being arbitrary} \\ \text{and } qp_i \text{ being reduced by } B(X) \end{array} \end{array}$$

These rules model the properties of an ideal, given in Def. 6. As for verification we do not explictely write down proof rules when reducing a boolean value constraint. In addition we extend proof rules by a deletion information, similar to clause deletion in [7]. We changed the proof checker PacTrim accordingly. Because of Thm. 1 and Thm. 2 the soundness and completeness arguments given in [23] can be generalized to polynomial rings over commutative rings with unity.

Our tool PacTrim validates the proof that the simplified AIG fulfills the given specification $\mathcal{L}$ by checking that $\mathcal{L}$ is derived and the derivation only uses valid proof rules. In addition we also check with DRAT-trim [24] the proofs generated by CaDiCaL for the CNF miter.

VII. Experiments

In our experiments we used an Intel Xeon E5-2620 v4 CPU at 2.10GHz (with turbo-mode disabled) with memory limit of 128 GB. The time is listed in rounded seconds (wall-clock time) and we measure the time from starting the tools until the tools are finished or we reach the time or memory limit. The source code, benchmarks and experimental data are available at http://fmv.jku.at/amulet.

In our experiments we aim to provide the most comprehensive comparison by considering all different multiplier architectures used in the current state-of-the-art [18]. These benchmarks are generated with the Arithmetic Module Generator [10], which allows to generate signed and unsigned integer multipliers up to bit-width 64. We only have access to truncated multipliers using SMT models, which we generated with Boolector [20]. Additionally we generated benchmarks of large multipliers with GenMul [19] (which only scales up to 512 bits), Boolector [20] and generator scripts by Arist Kojevnikov [9]. The multiplier architectures of [9], [20] are very simple architectures without any optimizations.

In the experiments presented in Table I we verify and certify different unsigned (u), signed (s) and truncated (t) multiplier architectures of 64 input bitwidth. Due to shortage of space we do not present experiments of smaller bitwidths. The time out for the experiments in this table is set to 3600 sec (1h).

We show the effect of our contributions by either omitting adder substitution and using only polynomial reduction for verification ("nosub"), omitting variable elimination ("noelim") or using the polynomial ring $\mathbb{Z}[X]$ instead of $\mathbb{Z}_l[X]$ ("nomod"). Each of the optimizations has a large effect and nearly every multiplier architecture, despite of the clean multiplier architecture "sp-ar-rc", produces a time out in one of the three columns. For truncated multipliers we would get a wrong result for "nomod", which is marked by "NA_3".

In the block "Verify" we measure the time for applying the tool chain as shown in the left side of Fig. 2. We list the times AMulet needs for adder substitution ("sub") and for verifying ("aig") as well as the time CaDiCaL uses to verify the CNF miter ("cnf"). Column "tot" lists the total time.

We compare our verification results to the most recent related works [18], [5], [22]. We want to highlight that the tool of [18] is not yet available, but it enhances the approach of [17]. Thus we list the experiments of their work [18], which are run on an Intel Xeon E3-1270 v3 CPU with 3.50 GHz and thus is a slightly faster CPU than ours. Experiments which are not available for comparison are marked by "NA_2".

The tool of [5] uses a certain optimization "&atree". After contacting the first and last authors of [28] we were told that this option only works for simple multipliers. Using this flag on more complex multipliers leads to incompleteness, which we mark again by "NA_3". If "&atree" is omitted, all experiments produce a segmentation fault.

It can be seen that in contrast to our previous work [22], we are able to verify all benchmarks within seconds and we are an order of magnitude faster than the currently most successful approach of [18]. The tools of related work are only partially applicable to verify signed and truncated multipliers, because the specification used in these tools is fixed to (un)signed multiplier circuits. We mark non-applicability with "NA_1".

TABLE I
VERIFICATION AND CERTIFICATION TIME

$\mathcal{L}$ column: unsigned (u), signed (s) or truncated (t) multiplier specification.

architecture	n	$\mathcal{L}$	nosub	nomod	noelim	Verify				[18]	[5]	[22]	Certify				Check			total	proof size	
						sub	cnf	aig	tot				sub	cnf	aig	tot	cnf	aig	tot		cnf	aig
sp-ar-rc	64	u	1	1	2	0	0	1	1	NA_2	0	133	0	0	2	2	0	3	3	5	0	188 290
sp-dt-lf	64	u	TO	1	3	0	0	2	2	31	NA_3	TO	0	0	2	3	0	3	3	6	34 423	186 170
sp-wt-cl	64	u	TO	TO	3	0	9	1	11	96	NA_3	TO	0	9	2	12	7	3	10	21	264 471	191 623
sp-bd-ks	64	u	TO	TO	2	0	1	1	3	162	NA_3	TO	0	2	2	4	1	3	4	8	78 567	190 915
sp-ar-ck	64	u	TO	1	2	0	0	1	1	143	NA_3	TO	0	0	2	2	0	3	3	5	1 432	187 251
bp-ar-rc	64	u	1	TO	118	0	0	1	1	53	NA_3	TO	0	0	2	2	0	3	3	5	0	161 815
bp-ct-bk	64	u	TO	TO	100	0	0	1	2	119	NA_3	TO	0	0	2	2	0	3	3	5	27 552	138 179
bp-os-cu	64	u	2	TO	TO	0	0	2	2	95	NA_3	TO	0	0	3	3	0	4	4	7	0	166 967
bp-wt-cs	64	u	1	TO	114	0	0	1	1	75	NA_3	TO	0	0	2	2	0	3	3	6	0	161 747
sp-ar-rc	64	s	1	1	2	0	0	1	1	NA_1	0	NA_1	0	0	2	2	0	3	3	6	0	188 426
bp-wt-cl	64	s	TO	3	109	0	10	1	11	NA_1	NA_3	NA_1	0	10	2	12	7	3	10	22	261 650	151 355
btor	64	t	0	NA_3	1	0	0	0	1	NA_1	NA_1	NA_1	0	0	1	1	0	1	1	2	0	70 374

NA_1: tool not applicable to type $\mathcal{L}$ NA_2: tool not yet available NA_3: incompleteness (see text) TO: 3600 sec

Benchmarks are either generated by the Arithmetic Module Generator of [10] or by Boolector [20] (btor).

PPG: simple (sp), Booth (bp) PPA: Dadda tree (dt), Wallace tree (wt), balanced delay tree (bd), array (ar), compressor tree (ct), overturned-stairs tree (os)
FSA: Ladner-Fischer (lf), carry look-ahead (cl), Kogge-Stone (ks), carry-skip (ck), ripple-carry (rc), Brent-Kung (bk), conditional sum (cu), carry select (cs)

In "Certify" and "Check" we present the time used for certifying and checking, cf. right side of Fig. 2. The columns of "Certify" have the same form as "Verify". In "Check" we list the times DRAT-TRIM ("cnf") [24] and PACTRIM ("aig") [23] need for proof checking. The column "total" lists the total time used to certify and check the multipliers. Certifying a multiplier is around twice as slow than verifying, because additional polynomial operations are necessary to match the proof rules. In the last two columns we present the sizes of the proofs. The proof size of CNFs is measured by the number of added RUP clauses [8]. A size of 0 means, that the final stage adder is not a GP adder. Thus a trivial CNF is reported which does not yield a resolution proof. The size of the algebraic proofs is measured by the number of PAC rules [23].

In the experiments of Table II we list the time to verify large multiplier designs. We are able to verify multipliers of input size 2048, consisting of more than 50 million AIG nodes in around 19h. Certifying and checking these benchmarks is around three times slower. For example certifying "kojvnkv-2048" needs 34h wall-clock time. Checking the (uncompressed) proof, which has a size of 1.4 TB, needs 20h.

VIII. CONCLUSION

In this paper we combine SAT and computer algebra to verify large unsigned, signed and truncated integer multipliers. Our theory describes polynomial reasoning over more general rings. We formulate and prove soundness and completeness.

We show how modular reasoning can be simulated by integer reasoning and revisit and apply existing D-Gröbner bases theory from the literature. Modular arithmetic is required to specify truncated multipliers. It also improves performance substantially. We formalize variable elimination too.

Our main contribution consists of extracting complex final stage adders, which are substituted by simple adders. Correctness of this substitution is proven by SAT and correctness of the simplified multiplier by our dedicated reduction engine.

TABLE II
VERIFYING BENCHMARKS OF LARGE INPUT SIZE

$\mathcal{L}$ column: unsigned (u) multiplier specification.

architecture	n	$\mathcal{L}$	Verify				[18]	[5]	AIG size
			sub	cnf	aig	tot			
btor	128	u	0	0	9	10	NA_2	2	123 k
kjvnkv	128	u	0	0	9	9	NA_2	2	195 k
sp-ar-rc	128	u	0	0	10	10	349	2	195 k
sp-dt-lf	128	u	0	2	13	15	490	NA_3	195 k
sp-wt-bk	128	u	0	1	18	20	746	NA_3	198 k
btor	256	u	1	0	119	120	NA_2	19	522 k
kjvnkv	256	u	1	0	84	86	NA_2	18	782 k
sp-ar-rc	256	u	1	0	84	86	8 720	20	782 k
sp-dt-lf	256	u	3	6	164	174	12 874	NA_3	780 k
sp-wt-bk	256	u	3	3	170	177	21 454	NA_3	790 k
btor	512	u	7	0	968	975	NA_2	300	2 093 k
kjvnkv	512	u	9	0	774	783	NA_2	247	3 138 k
sp-ar-rc	512	u	10	0	770	780	192 640	312	3 138 k
sp-dt-lf	512	u	25	21	1 539	1 585	240 051	NA_3	3 133 k
sp-wt-bk	512	u	24	9	1 560	1 594	492 320	NA_3	3 157 k
btor	1024	u	97	0	10 623	10 720	NA_2	8 323	8 379 k
kjvnkv	1024	u	106	0	5 463	5 570	NA_2	3 778	12 567 k
btor	2048	u	1 026	0	89 565	90 591	NA_2	150 976	33 536 k
kojvnkv	2048	u	1 057	0	67 733	68 790	NA_2	74 514	50 299 k

NA_2: tool not yet available NA_3: incompleteness (see text)

Benchmarks generated by Boolector [20] (btor), from [9] (kjvnkv) and [19].

PPG: simple (sp) PPA: array (ar), Dadda tree (dt), Wallace tree (wt)
FSA: ripple-carry (rc), Ladner-Fischer (lf), Brent-Kung (bk)

Our experiments show that the combination of these ideas allow to scale up verification to large multipliers of 2048 bits. We are also able to verify complex multipliers an order of magnitude faster than the previous state-of-the-art. Furthermore, we produce proof certificates in contrast to other approaches. These proofs are checked independently to validate the verification results.

In future work we want to apply our approach to synthesized multipliers where technology mapping is applied and to other arithmetic circuits beyond integer multipliers. Another intriguing research direction is to integrate both polynomial and clausal reasoning in a common proof format.

References

[1] M. Artin. *Algebra*. Prentice Hall, 1991.

[2] T. Becker, V. Weispfenning, and H. Kredel. *Gröbner Bases*. Springer, 1993.

[3] A. Biere. CaDiCaL at the SAT Race 2019. In *Proc. of SAT Race 2019*, 2019. Submitted.

[4] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[5] M. Ciesielski, T. Su, A. Yasin, and C. Yu. Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model. *IEEE TCAD*, pages 1–1, 2019.

[6] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 4-1-0. http://www.singular.uni-kl.de, 2016.

[7] M. Heule, W. A. H. Jr., and N. Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2013*, pages 181–188. IEEE, 2013.

[8] M. J. H. Heule and A. Biere. Proofs for Satisfiability Problems. In *All about Proofs, Proofs for All*, volume 55, pages 1–22, 2015.

[9] E. Hirsch, D. Itsykson, A. Kojevnikov, E. Kulikov, and S. Nikolenko. Report on the Mixed Boolean-Algebraic Solver 1. *Tech. Rep.*, 01 2005.

[10] N. Homma, Y. Watanabe, T. Aoki, and T. Higuchi. Formal Design of Arithmetic Circuits Based on Arithmetic Description Language. *IEICE Transactions*, 89-A(12):3500–3509, 2006.

[11] W. A. Hunt, M. Kaufmann, J. Strother Moore, and A. Slobodova. Industrial hardware and software verification with acl2. *Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences*, 375:20150399, 10 2017.

[12] D. Kaufmann, A. Biere, and M. Kauers. Incremental Column-wise verification of arithmetic circuits using computer algebra. *Formal Methods in System Design*, Feb 2019.

[13] D. Kaufmann, M. Kauers, A. Biere, and D. Cok. Arithmetic Verification Problems Submitted to the SAT Race 2019. In *Proc. of SAT Race 2019*, 2019. Submitted.

[14] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE TCAD*, 21(12):1377–1394, 2002.

[15] D. Lichtblau. Effective computation of strong Gröbner bases over Euclidean domains. *Illinois Journal of Mathematics*, 56, 11 2013.

[16] J. Lv, P. Kalla, and F. Enescu. Efficient Gröbner Basis Reductions for Formal Verification of Galois Field Arithmetic Circuits. *IEEE TCAD*, 32(9):1409–1420, 2013.

[17] A. Mahzoon, D. Große, and R. Drechsler. PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers. In I. Bahar, editor, *ICCAD*, page 129. ACM, 2018.

[18] 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 Conf.*, 2019. In press.

[19] A. Mahzoon, D. Große, and R. Drechsler. Multiplier Generator GenMul. http://www.sca-verification.org/, 2019.

[20] A. Niemetz, M. Preiner, C. Wolf, and A. Biere. Btor2 , BtorMC and Boolector 3.0. In *Computer Aided Verification, CAV*, volume 10981 of *LNCS*, pages 587–595. Springer, 2018.

[21] B. Parhami. *Computer Arithmetic - Algorithms and Hardware designs*. Oxford University Press, 2000.

[22] D. Ritirc, A. Biere, and M. Kauers. Column-wise verification of multipliers using computer algebra. In D. Stewart and G. Weissenbacher, editors, *Formal Methods in Computer-Aided Design, FMCAD 2017*, pages 23–30. IEEE, 2017.

[23] D. Ritirc, A. Biere, and M. Kauers. A Practical Polynomial Calculus for Arithmetic Circuit Verification. In A. Bigatti and M. Brain, editors, *3rd International Workshop on Satisfiability Checking and Symbolic Computation (SC2'18)*, pages 61–76. CEUR-WS, 2018.

[24] N. Wetzler, M. Heule, and W. A. Hunt Jr. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In C. Sinz and U. Egly, editors, *Intl. Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *LNCS*, pages 422–429. Springer, 2014.

[25] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Greuel. An Algebraic Approach for Proving Data Correctness in Arithmetic Data Paths. In *Computer Aided Verification, CAV*, volume 5123 of *LNCS*, pages 473–486. Springer, 2008.

[26] Wolfram Research, Inc. Mathematica, 2016. Version 10.4.

[27] C. Yu and M. Ciesielski. Formal Analysis of Galois Field Arithmetic Circuits-Parallel Verification and Reverse Engineering. *IEEE TCAD*, 38(2):354–365, Feb 2019.

[28] C. Yu, M. J. Ciesielski, and A. Mishchenko. Fast Algebraic Rewriting Based on And-Inverter Graphs. *IEEE TCAD*, 37(9):1907–1911, 2018.