Guess and Prove: A Hybrid Approach to Linear **Polynomial Recovery in Circuit Verification**

Clemens Hofstadler \square

Johannes Kepler University, Linz, Austria

Daniela Kaufmann 🖂 🗅

TU Wien, Vienna, Austria

– Abstract -

Formal verification of arithmetic circuits using computer algebra has been shown to be highly 8 successful. The circuit is encoded as a system of polynomials, which automatically generates a 9 lexicographic Gröbner basis. Correctness is then verified by computing the polynomial remainder of 10 the specification. To optimize the remainder computation, prior work extracts linear polynomials. 11 However, this required recomputing a Gröbner basis with respect to a degree-compatible order. 12

In this paper, we show that this computationally expensive step is unnecessary and propose 13 a novel hybrid verification approach that combines an FGLM-style linearization technique with a 14

guess-and-prove method using SAT solving to derive the linear relations directly from lexicographic 15

Gröbner bases. We enhance our approach using caching techniques and propagating vanishing 16

monomials. Our experimental results demonstrate that our method significantly outperforms previous 17

- linearization techniques. 18
- 2012 ACM Subject Classification Theory of Computation \rightarrow Automated Reasoning 19
- Keywords and phrases Computer Algebra, FGLM, And-Inverter Graphs, Hardware Verification 20
- Digital Object Identifier 10.4230/LIPIcs.CP.2025.23 21
- Supplementary Material Tool 22
- TALISMAN (Source Code): https://github.com/d-kfmnn/talisman/tree/be8187d 23
- Funding Clemens Hofstadler: LIT AI Lab funded by the state of Upper Austria 24
- Daniela Kaufmann: Austrian Science Fund (FWF) [10.55776/ESP666] 25
- Acknowledgements We thank Jérémy Berthomieu and the reviewers for insightful comments. 26

1 Introduction 27

Computer algebra-based formal verification techniques have proven to be highly effective in 28 verifying gate-level arithmetic circuits. As digital systems become more complex, ensuring 29 the accuracy of arithmetic circuits is essential, particularly in safety-critical fields like crypto-30 graphy and signal processing, where even minor errors can lead to significant consequences. 31 Formal verification approaches on the word-level based on theorem provers [24] or computer 32 algebra [12, 14, 16, 17], particularly those leveraging Gröbner bases [4], have demonstrated 33 significant advancements in recent years. 34

These methods encode circuits given as and-inverter graphs (AIGs) [18] as polynomial 35 systems where the coefficients are integers and the variables represent Boolean values. The 36 terms are sorted according to a reverse topological term order [19]. This ensures that the 37 output variable of a gate precedes its input variables, which results in an encoding where all 38 the leading terms are disjoint. Hence, the set of gate polynomials automatically generates a 39 Gröbner basis with respect to a lexicographic term order. This fact admits verification to be 40 performed by simply computing the remainder of the specification polynomial modulo the 41 Gröbner basis. The circuit is correct if and only if the final remainder is zero [15]. 42



© Clemens Hofstadler and Daniela Kaufmann.

licensed under Creative Commons License CC-BY 4.0 31st International Conference on Principles and Practice of Constraint Programming (CP 2025).

Editor: Maria Garcia de la Banda; Article No. 23; pp. 23:1–23:22 Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

23:2 Guess and Prove

However, a major bottleneck of using a lexicographic term order is the significant growth
of intermediate reduction results. This phenomenon, known as *monomial blow-up*, occurs
because the tails of the polynomials in the lexicographic Gröbner basis have higher degrees
than their leading terms, and can easily lead to intermediate reduction results with millions
of monomials [20]. To address this challenge, various preprocessing and rewriting algorithms
have been developed, which syntactically or semantically analyze the input circuit to remove
redundant information and choose a suitable reduction order.

Several advanced reduction engines that use a lexicographic order have been developed, 50 including DYNPHASEORDEROPT [16], DYPOSUB [21], and AMULET2 [13, 14], along with 51 its variant TELUMA [11]. The work in [13,14] employs SAT solving to rewrite parts of the 52 multiplier before applying an incremental column-wise verification algorithm. A subsequent 53 refinement in [11] eliminated the external SAT solver by using a sophisticated algebraic 54 encoding that includes polarities of literals. In [21], a dynamic rewriting approach is used 55 that determines the reduction order on-the-fly and backtracks if the size of the intermediate 56 reductions exceeds a predefined threshold. Building on this, [16] introduced an improved 57 method that incorporates mixed signals into the encoding. 58

Recent work [12] diverges from the lexicographic order and opts instead for a de-59 gree-compatible order to ensure that the degree of intermediate reduction results cannot 60 increase during reduction. By first linearizing the specification polynomial, the entire re-61 duction is restricted to linear polynomials. However, this approach requires extracting 62 linear polynomials from the circuit, which is not straightforward. The authors proposed 63 an on-the-fly technique that identifies subcircuits and computes degree-compatible Gröbner 64 bases only for them. The required linear polynomials for reduction are supposed to be 65 contained in the individually computed Gröbner bases. 66

Despite its innovation, the approach of [12] has notable limitations. First, it disregards 67 that the polynomials already form a lexicographic Gröbner basis, thereby missing important 68 structural advantages. Second, the method relies on an off-the-shelf computer algebra 69 system (CAS), which requires explicitly encoding that all variables are Boolean, missing 70 71 out on optimizations provided by modified polynomial operations. Third, whenever the initial subcircuit selection is inadequate, the subcircuit must be iteratively expanded, leading 72 to repeated incremental Gröbner basis computations. Since CASs cannot recognize that 73 parts of the input already generate a partial Gröbner basis, expensive computations must be 74 repeated. Our work aims to address these shortcomings. 75

⁷⁶ In this work, we introduce a novel method for extracting linear polynomials from sub-⁷⁷ circuits. Our hybrid approach combines an *FGLM-based linear extraction* algorithm with a ⁷⁸ guess-and-prove-style linearization technique.

In the *FGLM*-based algorithm, we initially compute the normal forms of single variables 79 with respect to the lexicographic Gröbner basis. The linear polynomials are then recovered by 80 determining the kernel of the corresponding coefficient matrix, which ensures that the linear 81 extractions are *proven* to be correct. This algorithm can be considered as the first steps of 82 the change-of-order FGLM-algorithm [8] for converting a Gröbner basis w.r.t. one term order 83 into one for a different order. The quess-and-prove linearization approach involves sampling 84 input values for the subcircuit. By solving a linear system of equations generated from these 85 samples, we can identify potentially valid linear polynomials. We prove the correctness of 86 the quessed polynomials using satisfiability (SAT) solving. To repair incorrect guesses, we 87 exploit information obtained from the SAT solving process. 88

⁸⁹ Despite taking advantage of properties that are characteristic for circuit verification, we ⁹⁰ highlight that our methods are not restricted to this application and can be used in any

⁹¹ context that meets these characteristics. Therefore, we introduce the FGLM and guess-and-

prove algorithms in a general setting in Section 3, before discussing how to specialize them
 for circuit verification in Section 4.

We further enhance our method by reusing computations for isomorphic subcircuits, and by generating and propagating vanishing monomials, see Section 4.1. We implement our techniques in a new tool, called TALISMAN. Our experimental evaluation (Section 5) shows that we significantly outperform existing linearization techniques.

98 **2** Preliminaries

⁹⁹ In Section 2.1, we recall basic concepts about polynomials and Gröbner bases that will be ¹⁰⁰ needed for the rest of this work. For further information, we refer to the great introductory ¹⁰¹ books [5,6], which also served as basis for our presentation. Section 2.2 introduces AIGs and ¹⁰² how they can be encoded into polynomials.

2.1 Polynomials and Gröbner Bases

¹⁰⁴ Fix a finite set $X = \{x_1, \ldots, x_n\}$ of indeterminates and let \mathbb{K} be a field.

▶ Definition 1 (Monomials). A monomial (in X) is a product of the form $x_1^{\alpha_1} \dots x_n^{\alpha_n}$ with exponents $\alpha_1, \dots, \alpha_n \in \mathbb{N}$. The set of all monomials in X is denoted by [X]. The (total) degree deg(m) of a monomial $m = x_1^{\alpha_1} \dots x_n^{\alpha_n} \in [X]$ is the sum of exponents, i.e., deg(m) = $\alpha_1 + \dots + \alpha_n$.

▶ Definition 2 (Polynomials). A polynomial f (over \mathbb{K}) is a finite \mathbb{K} -linear combination of monomials, that is, $f = c_1m_1 + \cdots + c_sm_s$, with coefficients $c_1, \ldots, c_s \in \mathbb{K}$ and $m_1, \ldots, m_s \in [X]$. The set of all polynomials is denoted by $\mathbb{K}[X] = \mathbb{K}[x_1, \ldots, x_n]$. A polynomial f is linear if it is of the form $f = c_0 + c_1x_1 + \cdots + c_nx_n$ with $c_0, \ldots, c_n \in \mathbb{K}$.

For a polynomial $f = c_1m_1 + \cdots + c_sm_s \in \mathbb{K}[X]$, we call the product c_im_i a term of f, for $i = 1, \ldots s$. In this work, we consider all polynomials to be in canonical form, which means that terms with equal monomials are merged by adding their coefficients and terms with coefficient zero are omitted. Furthermore, as we will only consider polynomial equations with right hand side zero, we will also write "f" instead of "f = 0".

Many algebraic operations (such as division) require the terms within a polynomial to be sorted. This is achieved by a *monomial order*.

- **Definition 3** (Monomial Order). A total order \prec on [X] is a monomial order if it satisfies:
- 121 **1.** $m_1 \prec m_2$ implies $m \cdot m_1 \prec m \cdot m_2$ for all $m, m_1, m_2 \in [X]$;
- 122 **2.** every nonempty subset of [X] has a smallest element;

Example 4. We list two classical examples of monomial orders. Let $m_1 = x_1^{\alpha_1} \dots x_n^{\alpha_n}$ and $m_2 = x_1^{\beta_1} \dots x_n^{\beta_n}$ be monomials.

1. Lexicographic order: We say that $m_1 \prec_{\text{lex}} m_2$ if $\alpha_i < \beta_i$ for the smallest index $i \in \{1, \ldots, n\}$ where $\alpha_i \neq \beta_i$.

127 **2.** Degree lexicographic order: We say that $m_1 \prec_{\text{dlex}} m_2$ if $\deg(m_1) < \deg(m_2)$ or if 128 $\deg(m_1) = \deg(m_2)$ and $m_1 \prec_{\text{lex}} m_2$.

The degree lexicographic order is an instance of a *degree-compatible* (or *graded*) order.

A monomial order \prec is *degree-compatible* if deg $(m_1) < \text{deg}(m_2)$ implies $m_1 \prec m_2$ for all monomials $m_1, m_2 \in [X]$. With respect to a fixed monomial order \prec , we can identify the unique maximal monomial in each nonzero polynomial $f \in \mathbb{K}[X]$. This monomial is called the *leading monomial* of fand denoted by $\operatorname{Im}(f)$. The coefficient of $\operatorname{Im}(f)$ is called the *leading coefficient* of f, denoted by $\operatorname{lc}(f)$, and the term $\operatorname{lc}(f) \operatorname{Im}(f)$ is called the *leading term* of f, denoted by $\operatorname{lt}(f)$. The *tail* of f is $\operatorname{tail}(f) = f - \operatorname{lt}(f)$.

▶ Definition 5 (Ideal). A nonempty subset $I \subseteq \mathbb{K}[X]$ is an ideal if it is closed under addition, that is, $f + g \in I$ for all $f, g \in I$, and if it is closed under multiplication by arbitrary polynomials, that is, $hf \in I$ for all $f \in I$ and $h \in \mathbb{K}[X]$.

A set of polynomials $F = \{f_1, \ldots, f_r\} \subseteq \mathbb{K}[X]$ can be considered as a system of equations f_{141} $f_1 = \cdots = f_r = 0$. The set of all polynomials h for which the equation h = 0 can be derived algebraically from this system (by adding equations and by multiplying an equation by a polynomial) forms an ideal, which is denoted by $\langle F \rangle$. It is given explicitly by $\langle F \rangle = \{h_1 f_1 + \cdots + h_r f_r \mid h_1, \ldots, h_r \in \mathbb{K}[X]\}.$

¹⁴⁵ A central result in (computational) algebra, known as *Hilbert's Basis Theorem* [5, ¹⁴⁶ Thm. 2.5.4], says that every ideal $I \subseteq \mathbb{K}[X]$ can be written as $I = \langle F \rangle$ for some finite ¹⁴⁷ set $F \subseteq \mathbb{K}[X]$. The set F is called a *basis* of I and we say that I is generated by F.

In general, an ideal has many bases. Given an arbitrary basis F of an ideal, it is a difficult 148 question to determine whether an equation h = 0 can be derived from the system induced 149 by F, or in algebraic terms, to determine whether $h \in \langle F \rangle$. The latter problem is known 150 as the *ideal membership problem*. In principle, it can be solved by repeatedly dividing h by 151 the elements in the basis F until a remainder is obtained that cannot be divided further. 152 See [5, Thm. 2.3.3] for a precise description of this division process. However, this remainder 153 is typically not unique but it depends on the order of divisions. Only for certain particular 154 bases of an ideal a unique remainder can be obtained. These bases are called *Gröbner bases* 155 and they can be used to solve the ideal membership problem. The following definition 156 depends on a monomial order; we assume that some order \prec has been fixed. 157

▶ Definition 6 (Gröbner Basis). A basis G of an ideal $I \subseteq \mathbb{K}[X]$ is a Gröbner basis of I (w.r.t. \prec) if every polynomial $f \in \mathbb{K}[X]$ has a unique remainder under division by G. This unique remainder is denoted by NF_G(f) and is also called the normal form of f.

An equivalent characterisation of a Gröbner basis G of I is that $f \in I$ if and only if NF_G(f) = 0, see [5, Cor. 3.6.2]. Moreover, it can be shown that every ideal has a finite Gröbner basis. A Gröbner basis of an ideal I can be computed starting from any finite basis of I by a completion procedure that can be considered as an algebraic analogue of the Knuth–Bendix completion algorithm. This completion procedure is known as *Buchberger's algorithm* [4] and, in contrast to Knuth–Bendix, is guaranteed to terminate for any input.

A polynomial $f \in \mathbb{K}[X] = \mathbb{K}[x_1, \ldots, x_n]$ vanishes on a set of points $P \subseteq \mathbb{K}^n$ if $f(a_1, \ldots, a_n) = 0$ for all $(a_1, \ldots, a_n) \in P$. For an ideal $I \subseteq \mathbb{K}[X]$, the set of all points on which all polynomials in I vanish is called the *(algebraic) variety* of I, denoted by V(I), i.e., $V(I) = \{(a_1, \ldots, a_n) \in \mathbb{K}^n \mid f(a_1, \ldots, a_n) = 0$ for all $f \in I\}$. If V(I) is finite, then I is called zero-dimensional.

2.2 Polynomial Encodings of And-Inverter Graphs

An AIG [18] is a directed acyclic graph that can be used to represent Boolean functions and logic circuits in a compact way. The nodes (also called *gates*) in an AIG correspond to logical conjunction. The edges represent signal propagation, with optional markers indicating negation. The *primary inputs* of the AIG represent Boolean variables.

184





▶ Definition 7 (Specification). The specification of an AIG is a polynomial $S \in \mathbb{K}[X]$ that relates the outputs of an AIG to its primary inputs.

¹⁷⁹ While the AIG performs logical operations on Boolean variables, the specification is not ¹⁸⁰ confined to the Boolean ring $\mathbb{B}[X]$. Instead, it can extend to alternative coefficient domains, ¹⁸¹ such as $\mathbb{Q}[X]$, depending on the underlying AIG.

▶ Definition 8 (Gate Polynomials G(C)). We map $\top \mapsto 1$ and $\bot \mapsto 0$ to derive the relations a $\land b \cong ab$ and $\neg a \cong 1 - a$ between logic and algebra. Let g be an AIG node with inputs a, b:

$Logical \ constraint$		Gate polynomial
$g = a \wedge b$	\Rightarrow	g-ab
$g = \neg a \wedge b$	\Rightarrow	g - (1 - a)b = g + ab - b
$g = a \wedge \neg b$	\Rightarrow	g - a(1 - b) = g + ab - a
$g = \neg a \land \neg b$	\Rightarrow	g - (1 - a)(1 - b) = g - ab + b + a - 1

For a given AIG C, we collect all its gate polynomials in the set G(C).

Since the specification is not restricted to the Boolean ring, we need to add further polynomials that restrict the variables to the Boolean domain.

Definition 9 (Boolean Value Polynomials B(C)). For every primary input a_i of the AIG we define a Boolean value polynomial $a_i(a_i - 1) = a_i^2 - a_i = 0$ that encodes that the variable can only take the values 0 and 1. Let B(C) denote the set of Boolean value polynomials.

It suffices to define the Boolean value polynomials only for the primary inputs, as they propagate, i.e., $\forall x_i \in X : x_i^2 - x_i \in \langle G(C) \cup B(C) \rangle$, with X being the set of all output, gate, and input variables [15].

Example 10. Figure 1 shows an AIG representing a 2-bit multiplier and its corresponding polynomial encoding. We denote the primary inputs by a_0, a_1, b_0, b_1 and outputs by s_0, s_1, s_2, s_3 . The internal nodes are denoted by g_i , where *i* corresponds to the respective AIG node. The specification S expresses that the weighted sum of output bits (= output bit-vector) is equal to the product of the weighted sum of input bits (= input bit-vectors).

¹⁹⁹ The correctness of the AIG can be shown by deriving that S is implied by the polynomial ²⁰⁰ encoding, which algebraically means that we want to show that $S \in \langle G(C) \cup B(C) \rangle$ [15]. 201 Since the terms are sorted lexicographically according to a reverse topological variable order,

²⁰² all leading terms are disjoint. Hence the polynomials automatically generate a Gröbner basis.

Thus, verification can be performed by computing the remainder of S modulo $G(C) \cup B(C)$.

²⁰⁴ The circuit is correct if and only if the final remainder is zero [15].

²⁰⁵ **3** Finding Linear Polynomials in Ideals

A key step in our approach to circuit verification is identifying linear polynomials in a given ideal. In this section, we propose two methods that recover linear polynomials by exploiting essential properties of the ideals in our application.

We highlight that, while these techniques are designed to take advantage of these properties, they are not specific to our application and can be used in any context that meets our assumptions about the ideals involved. Formally, we consider the following problem over a (computable) field K:

213Given: an ideal $I \subseteq \mathbb{K}[X]$ 214Compute: the set $\mathbb{L}(I)$ of all linear polynomials in I

Note that $\mathbb{L}(I)$ is non-empty because it always contains the zero polynomial. Moreover, if $p, q \in \mathbb{L}(I)$, then so are p + q and cp for every $c \in \mathbb{K}$. Thus, we obtain the following result.

▶ Lemma 11. The set $\mathbb{L}(I)$ of all linear polynomials of an ideal $I \subseteq \mathbb{K}[X]$ forms a finitedimensional vector space over \mathbb{K} of dimension at most |X| + 1.

Proof. We have already seen that $\mathbb{L}(I)$ is a vector space. The dimension result follows from the fact that $\mathbb{L}(I)$ is a subspace of the (|X| + 1)-dimensional vector space $\mathbb{L}(\mathbb{K}[X])$, which has the basis $X \cup \{1\}$.

By "computing" the set $\mathbb{L}(I)$, we mean determining a K-vector space basis. In the general case, when the ideal I is given by an arbitrary basis without additional structural properties, the only approach we are aware of to obtain this basis for $\mathbb{L}(I)$ is by computing a Gröbner basis of I w.r.t. a degree-compatible monomial order. By [12, Thm. 1], such a Gröbner basis contains the desired basis as a subset. This was also the approach used in [12]. However, in the application of circuit verification, the given basis has a lot more structure:

1. It forms a Gröbner basis w.r.t. a lexicographic monomial order.

229 2. The lexicographic Gröbner basis enables easy computation (of a subset) of V(I).

A naïve computation of a Gröbner basis w.r.t. a degree-compatible order using a black-box
 computation completely ignores this existing structure.

232 3.1 FGLM-style Linearization

²³³ While the lexicographic order is not degree-compatible, we can still exploit the fact that ²³⁴ we already have a Gröbner basis w.r.t. this order when computing a basis of $\mathbb{L}(I)$. More ²³⁵ generally, the method presented in this section works with arbitrary Gröbner bases. Thus, ²³⁶ we consider the following refined problem:

Given: an ideal $I \subseteq \mathbb{K}[X]$ and a Gröbner basis G of ICompute: the set $\mathbb{L}(I)$ of all linear polynomials in I

Algorithm 1 FGLM-style-Linearization

Input : Gröbner basis G of an ideal $I \subseteq \mathbb{K}[X] = \mathbb{K}[x_1, \dots, x_n]$ **Output**: a \mathbb{K} -vector space basis L of $\mathbb{L}(I)$

- 1 Compute the normal forms $NF_G(1), NF_G(x_1), \ldots, NF_G(x_n);$
- 2 Set up the coefficient matrix

$$A \leftarrow \begin{pmatrix} | & | & | \\ \mathrm{NF}_G(1) & \mathrm{NF}_G(x_1) & \cdots & \mathrm{NF}_G(x_n) \\ | & | & | \end{pmatrix}$$

3 $b_1, \ldots, b_k \leftarrow$ a basis of ker(A); **4** return $L \leftarrow \{b_{i,0} + b_{i,1}x_1 + \cdots + b_{i,n}x_n \mid b_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,n}), i = 1, \ldots, k\};$

Our method is based on the classical fact that if G is a Gröbner basis of an ideal (w.r.t. any monomial order), then the normal form map $f \mapsto NF_G(f)$ is linear, i.e., $NF_G(cf + dg) = cNF_G(f) + dNF_G(g)$, for all $c, d \in \mathbb{K}$ and $f, g \in \mathbb{K}[X]$, cf. [5, Ex. 2.6.13]. We use this linearity to compute a basis of $\mathbb{L}(I)$.

▶ Lemma 12. Let G be a Gröbner basis of an ideal $I \subseteq \mathbb{K}[X] = \mathbb{K}[x_1, \ldots, x_n]$. A linear polynomial $f = c_0 + c_1x_1 + \cdots + c_nx_n$ with $c_0, \ldots, c_n \in \mathbb{K}$ is contained in I if and only if

$$c_0 \operatorname{NF}_G(1) + c_1 \operatorname{NF}_G(x_1) + \dots + c_n \operatorname{NF}_G(x_n) = 0.$$

Proof. Follows from the linearity of the map $f \mapsto NF_G(f)$ and the fact that $f \in I$ if and only if $NF_G(f) = 0$.

Lemma 12 outlines a procedure for computing a basis of $\mathbb{L}(I)$: First, compute the normal 245 forms $NF_G(1), NF_G(x_1), \ldots, NF_G(x_n)$ using the Gröbner basis G. Then find all K-linear 246 relations between these normal forms, which can be achieved as follows: Form the coefficient 247 matrix A whose columns contain the coefficients of $NF_G(1), NF_G(x_1), \ldots, NF_G(x_n)$. More 248 precisely, we assign to each row A_i of A a monomial m_i that appears in the normal forms. 249 The entry $A_{i,j}$ is then given by the coefficient of m_i in NF_G (x_j) (with $x_0 = 1$). With 250 this, a basis of the kernel ker(A) of A yields a basis of $\mathbb{L}(I)$ by translating any basis vector 251 $c = (c_0, c_1, \dots, c_n) \in \ker(A)$ into the linear polynomial $c_0 + c_1 x_1 + \dots + c_n x_n \in \mathbb{L}(I)$. 252 Recall that a basis of ker(A) can be computed by Gaussian elimination, which has a cubic 253 computational complexity in the matrix size; see, e.g., [23, Sec. 7.1]. These steps are 254 summarized in Algorithm 1. 255

Proposition 13. Algorithm 1 terminates and is correct.

Proof. Termination is clear. For correctness, note that Lemma 12 implies that there is a one-to-one correspondence between linear polynomials $c_0 + c_1 x_1 + \cdots + c_n x_n \in \mathbb{L}(I)$ and vectors $(c_0, c_1, \ldots, c_n) \in \ker(A)$. Thus, any basis of the latter can be translated into a basis of the former (and vice versa).

Example 14. Consider the ideal $I \subseteq \mathbb{Q}[a, b, g_1, g_2, g_3, g_4]$ generated by the four polynomials

 $g_{1} = ab,$ $g_{2} = (1-a)(1-b),$ $g_{3} = a(1-b),$ $g_{4} = (1-g_{1})(1-g_{2}),$

as well as by the two Boolean value polynomials $a^2 - a$ and $b^2 - b$. These six polynomials form a Gröbner basis of I w.r.t. the lexicographic order where $a \prec b \prec g_1 \prec g_2 \prec g_3 \prec g_4$.

23:8 Guess and Prove

We use Algorithm 1 to compute a Q-vector space basis of $\mathbb{L}(I)$. To this end, we compute the normal forms of all six variables and of 1 w.r.t. the given Gröbner basis and set up the corresponding coefficient matrix A. This yields

		N	$F_G(1)$	$NF_G(a)$	$NF_G(b)$	$NF_G(g_1)$	$NF_G(g_2)$	$NF_G(g_3)$	$NF_G(g_4)$	
		1	1	0	0	0	1	0	0	1
268	A =	[0	1	0	0	-1	1	1	a
			0	0	1	0	-1	0	1	b
			0	0	0	1	1	-1	-2	/ ab

The kernel of A is generated by the three vectors (0, -1, -1, 2, 0, 0, 1), (0, -1, 0, 1, 0, 1, 0), and (-1, 1, 1, -1, 1, 0, 0). Thus, a basis of $\mathbb{L}(I)$ is given by the three linear polynomials

 $g_4 + 2g_1 - a - b,$ $g_3 + g_1 - a,$ $g_2 - g_1 + a + b - 1.$

We call Algorithm 1 "FGLM-style linearization" because it can be considered as the first steps of the change-of-order FGLM-algorithm [8] for converting a Gröbner basis G w.r.t. one monomial order into another one G' for a different order.

Typically, the FGLM-algorithm is used to convert a degree-compatible Gröbner basis into 275 one for a lexicographic order, because the former are easier to compute while the latter are 276 better suited for solving polynomial systems. Nevertheless, the FGLM-algorithm can be used 277 to transition between any monomial orders as long as the underlying ideal is zero-dimensional. 278 Algorithm 1 can be considered as the first steps of an FGLM-computation for converting a 279 Gröbner basis into a Gröbner basis w.r.t. a degree-compatible order. We note that, while 280 FGLM is only applicable to zero-dimensional ideals, Algorithm 1 works for arbitrary ideals. 281 The complexity of Algorithm 1 is essentially cubic in the size of the computed normal forms 282 $NF_G(1), NF_G(x_1), \ldots, NF_G(x_n)$, which could be exponential in the worst case, cf. [10, Ex. 5.1]. 283 In our application, we found that Algorithm 1 works well for ideals generated by several dozen 284 polynomials in a few dozen variables. However, for dealing with more involved (sub)circuits, 285 we have to handle ideals spanned by several thousand polynomials in several thousand 286 variables. For such ideals, the normal forms consist of tens of millions of terms and are too 287 large to compute. Therefore, in these cases, we switch to a different method, exploiting the 288 fact that we can cheaply compute many points in the variety V(I) of the considered ideals. 289

3.2 Guess-and-Prove Linearization

In cases where we do not have access to a Gröbner basis or when computing the normal forms required in Algorithm 1 is prohibitively expensive, we can employ a different strategy: When (sufficiently many) points in the variety V(I) can be computed, we can guess candidates for linear polynomials by interpolation and then verify their correctness *a posteriori*. Our approach is based on the following basic result.

▶ Lemma 15. Let $I \subseteq \mathbb{K}[X]$ be an ideal and let $P \subseteq V(I)$. If $S \subseteq \mathbb{K}[X]$ is the set of all linear polynomials that vanish on P, then $\mathbb{L}(I) \subseteq S$.

²⁹⁸ **Proof.** Follows from the fact that all polynomials in I vanish on all points in V(I).

Lemma 15 provides a necessary condition for determining whether an ideal I contains any nonzero linear polynomials. Given a set of points $P \subseteq V(I)$, we can compute a K-vector space basis for the set S by solving a system of linear equations (see lines 3 – 6 in Algorithm 2). If this basis is empty, it follows that $\mathbb{L}(I) = \{0\}$. More precisely, if the computed basis does

Algorithm 2 Guess-And-Prove-style Linearization

: a zero-dimensional radical ideal $I \subseteq \mathbb{K}[X]$ over an algebraically closed field \mathbb{K} , Input methods sample and verify as specified in Sec. 3.2, and a positive integer $N \in \mathbb{N}_{>0}$ **Output**: a \mathbb{K} -vector space basis L of $\mathbb{L}(I)$ 1 $A \leftarrow \text{empty matrix};$ $\mathbf{2}$ while true do 3 $p_1, \ldots, p_N \leftarrow \text{use sample}(I) \text{ to compute } N \text{ points in } V(I);$ Prepend the coordinate 1 to each p_i and add these vectors as new rows to A, i.e., 4 $A \leftarrow \begin{pmatrix} A \\ 1 & -p_1 - \\ \vdots & \vdots \\ 1 & -p_N - \end{pmatrix}$ $b_1, \ldots, b_k \leftarrow$ a basis of ker(A); $\mathbf{5}$ $L \leftarrow \{b_{i,0} + b_{i,1}x_1 + \dots + b_{i,n}x_n \mid b_i = (b_{i,0}, b_{i,1}, \dots, b_{i,n}), i = 1, \dots, k\};$ 6 if $\forall f \in L : \texttt{verify}(f)$ then 7 return L; 8

not include a linear polynomial involving a given variable $x \in X$, then we can conclude that neither does $\mathbb{L}(I)$.

Obviously, the set S can contain wrong guesses and be (a lot) larger than $\mathbb{L}(I)$. To turn Lemma 15 into an actual algorithm for computing a basis of $\mathbb{L}(I)$, we have to make some additional assumptions:

1. The ideal I is zero-dimensional, i.e., V(I) is finite.

2. The ideal I is radical, which means that $f^r \in I$ implies $f \in I$, for all $f \in \mathbb{K}[X], r \in \mathbb{N}$.

 $_{310}$ 3. The coefficient field K is algebraically closed, which means that every univariate polynomial

in $\mathbb{K}[x]$ has a root in \mathbb{K} . For example, \mathbb{C} is algebraically closed, while \mathbb{Q} and \mathbb{R} are not.

In the worst case, we may need to process all points in V(I). The zero-dimensionality of Iensures that we can do this in finite time. The other two conditions exclude pathological cases where polynomials have roots with higher multiplicity or roots in an extension of K.

We furthermore assume that we have access to a function sample(I), which returns a point of V(I) and, when called sufficiently often, enumerates all of V(I). Additionally, a function verify(f) is assumed, which returns "true" if and only if $f \in I$ and "false" otherwise. We deliberately keep the functions sample and verify abstract in Algorithm 2 to allow for a high-level presentation. Their instantiation depends on the concrete application. We discuss their instantiation for our application of circuit verification in the following section.

Provided that an ideal I meets all the above assumptions, we can compute a basis of $\mathbb{L}(I)$ through an iterative process, summarized in Algorithm 2. First, we use sample(I) to enumerate a subset of points $P \subseteq V(I)$. We then compute a basis L of linear polynomials that vanish on P using linear algebra. Note that, to also recover linear polynomials with a constant term, we have to prepend an additional coordinate 1 to all points in P. If $L \subseteq I$, we have found a basis of $\mathbb{L}(I)$. Otherwise, we sample additional points from V(I) and repeat the process. The assumptions on I ensure that this procedure will eventually terminate.

Proposition 16. Algorithm 2 terminates and is correct.

Proof. For termination, note that the assumptions on the method sample ensure that the matrix A will eventually contain all of V(I) as its rows. Then, by construction, all elements

	Algorithm 3 Verification using linear extractions
	Input : Circuit C in AIG format, Specification polynomial S
	Output : Determine whether C fulfills the specification
1	$G_{\text{init}} \leftarrow \text{Polynomial-Encoding}(C);$
2	$S_{\text{lin}}, G_{\text{ext}} \leftarrow \text{Linearize-Spec-wrt-AIG}(S, G_{\text{init}});$
3	$G \leftarrow \operatorname{Preprocessing}(G_{\operatorname{ext}});$
4	while $\operatorname{lm}(\mathcal{S}_{\operatorname{lin}}) \in \{\operatorname{lm}(g) g \in G\}$ do
5	$p \leftarrow g \in G$ such that $\operatorname{lm}(g) = \operatorname{lm}(\mathcal{S}_{\operatorname{lin}}); G_{\operatorname{sub}} \leftarrow \emptyset;$
6	while $\nexists p_{\text{lin}}$ and Can-Increase (G_{sub}) do
7	$G_{\text{sub}} \leftarrow \text{Subcircuit}(\text{lm}(g), G, G_{\text{sub}});$
8	if $ G_{sub} $ is moderate then $L \leftarrow FGLM$ -style-Linerization (G_{sub}) ;
9	else $L \leftarrow \text{Guess-and-Prove}(G_{\text{sub}}, \text{AdBinSample}, \text{SAT}, \min(10 \cdot G_{\text{sub}} , 10^4));$
10	$p_{\text{lin}} \leftarrow p \in L \text{ such that } \text{lm}(p) = \text{lm}(\mathcal{S}_{\text{lin}});$
11	if $\nexists p_{\text{lin}}$ then return \bot ;
12	$\mathcal{S}_{\text{lin}} \leftarrow \text{Linear-Reduce}(\mathcal{S}_{\text{lin}}, p_{\text{lin}});$
13	return $S_{\text{lin}} = 0;$

in L vanish on all of V(I). With this, Hilbert's Nullstellensatz [5, Thm. 4.1.2] together with the assumption that I is radical ensure that $L \subseteq I$. Thus, the test in line 7 will succeed and the algorithm will terminate and return L. For correctness, the returned set L consists of linear polynomials that all lie in I by the test in line 7. Thus $L \subseteq \mathbb{L}(I)$, and Lemma 15 ensures that L is actually a basis of $\mathbb{L}(I)$.

In the worst case, Algorithm 2 has to consider all points in V(I) before finding a correct 336 basis of $\mathbb{L}(I)$. However, in practice, a much smaller subset of points is often sufficient, 337 especially if the sampling strategy is well-designed. In particular, Algorithm 2 can be 338 optimized by including a *repair step*, which prevents wrong guesses from reoccurring in 339 subsequent iterations. If verify(f) in line 7 fails for a polynomial $f \in L$, i.e., $f \notin I$, then 340 there is at least one point $p \in V(I)$ such that $f(p) \neq 0$. This point p cannot have been 341 sampled in line 3. If p can be computed, adding it to A in the next iteration ensures that 342 the incorrect guess f cannot reoccur, thereby improving the algorithm's convergence rate. 343

4 Recovering Linear Polynomials for Circuit Verification

We now discuss how we use the FGLM-style linearization and guess-and-prove-style linearization algorithms of the previous section in the context of multiplier verification.

Multipliers typically consist of three components: (1) partial product generation (PPG), where the individual bitwise products are computed, (2) partial product accumulation (PPA), where these products are combined using compression techniques such as Wallace trees or Dadda trees, and (3) the final-stage adder (FSA), which produces the final multiplication result. The FSA is particularly critical in determining circuit efficiency, and may use advanced addition techniques like carry-lookahead techniques to speed up computation.

The main loop of our approach to circuit verification is outlined in Algorithm 3. The structure of this loop follows the idea from [12] of linearizing only subcircuits instead of attempting to extract linear polynomials from the entire circuit. We discuss optimizations of Algorithm 3 in Section 4.1.

Line 1 - 2: We encode the circuit using the translation presented in Definitions 8 and 9, and linearize the specification polynomial $S \in \mathbb{K}[X]$ by replacing every non-linear monomial m_i

by a new variable y_i and adding the set $\Gamma = \{y_i - m_i \mid y_i \notin X \land m_i \in S \land \deg(m_i) > 1\}$ to the set of gate polynomials. The correctness proof for this step can be found in [12, Lem. 3]. *Line 3:* We apply the same basic preprocessing techniques as in [12]. That is, we eliminate all gates that only occur positively and detect vanishing monomials. Vanishing monomials are pairs of nodes that have the same children, but with different polarities. Thus, the product of the parent nodes is equal to zero. We additionally enhance the preprocessing by propagating vanishing monomials, see Section 4.1.

Furthermore, we identify and mark parts of the circuit that require dedicated reasoning. In the case of multiplier circuits, these correspond to the FSA. Whenever the FSA employs carry-lookahead techniques, we must treat the entire FSA as a single subcircuit to derive the required linear polynomials. This necessity arises because carry-lookahead techniques compute carries in parallel, unlike sequential approaches such as ripple-carry adders. For identifying these subcircuits, we utilize the cut identification method implemented in AMULET2 [14].

Line 4 - 6: The algorithm then iterates over the linearized specification as long as we have a gate polynomial in the encoding that has the same leading monomial as the linearized specification. In each iteration, we generate a subcircuit with root node g for which we perform the linearization. If no matching linearized polynomial p_{lin} is found, we increase the subcircuit size until completion. This is the case when the subcircuit contains all gates that are topologically smaller than g.

Line 7: We identify the subcircuit as follows. If the root node g contains a marking, i.e., if it belongs to a pattern identified in line 3, all nodes with the same marking are collected and returned. For unmarked nodes, the algorithm initially constructs a subcircuit G_{sub} by collecting all nodes up to a given depth d with a fanout size below the specified limit f_{out} . Our initial values are d = 2 and $f_{\text{out}} = 4$ as this has empirically shown to be the most efficient. Additionally, we include all nodes whose children are already part of G_{sub} . We also promote all input nodes of G_{sub} with a fanout of one to be included in G_{sub} .

Rather than immediately increasing the depth d in following iterations, we attempt to expand the subcircuit by adding individual input nodes of $G_{\rm sub}$. The expansion prioritizes the largest input node, based on our predefined variable ordering that still has a fanout size at most $f_{\rm out}$. However, every 15 iterations, the algorithm restarts with incremented depth and fanout limits. The threshold of 15 was empirically selected to keep the alternations between the depth-first and breath-first selection approaches balanced.

In that sense, our subcircuit selection algorithm is more sophisticated than the one in [12], which did not consider expanding the subcircuit by individual nodes, neither was there a limit on the fanouts nor were special markings considered.

Line 8: If G_{sub} is of moderate size, that is, the number of polynomials is less than 100, and the depth is at most 5 (which in our setting corresponds to all subcircuits except the marked FSA), we use the FGLM-based extraction method.

There is one exception: we also use the FGLM-based extraction method for FSA where, after preprocessing, we detect polynomials with a degree greater than the input bitwidth of the circuit, but consisting only of two terms. We detected that the guess-and-prove approach will mostly produce incorrect guesses for such polynomials due to the high-degree monomials evaluating to zero exponentially often and evaluating to 1 only for one particular input.

Line 9: For larger subcircuits with polynomials of moderate degree, we use the guessand-prove approach to avoid excessive growth in normal forms, which would cause the computation to stall in the FGLM-style algorithm.

First of all, let us note that the ideals arising from the subcircuits satisfy all assumptions made in Section 3.2. They are zero-dimensional and radical because they contain, by definition, the Boolean value polynomials for all variables. The presence of these polynomials also takes care of the third assumption: Even if a considered ideal I is formally defined over a field \mathbb{K} that is not algebraically closed, we can implicitly treat it as defined over the algebraic closure $\overline{\mathbb{K}}^1$ of \mathbb{K} . The Boolean value polynomials ensure that the underlying variety V(I)remains unchanged, which guarantees that our algorithm will return the correct result.

We instantiate the sample and verify plugins required in Algorithm 2 as follows. For 412 sampling, we apply an adaptive strategy. In the first iteration, we randomly draw values 0 413 and 1 for all the inputs of the subcircuit and propagate the signals according to the gate 414 structure of the subcircuit. We draw at least 10 times the size of gates in $G_{\rm sub}$ samples, 415 but at most 10000. This ensures that we have enough samples, while keeping the matrix A416 in Algorithm 2 at a computationally manageable size. To ensure that every input appears 417 with both polarities, we also include the trivial samples by setting all inputs to 0 and to 1, 418 respectively. If more than one iteration is required in Algorithm 2, we switch from random 419 sampling to the repair strategy discussed at the end of Section 3.2. Using our verification 420 plugin, we can compute witnesses for wrong guesses in the form of points in the variety and 421 only append those witnesses to the matrix in subsequent iterations. We call our strategy 422 adaptive-binary-sampling (AdBinSample). 423

As a verification technique, we use SAT solving (SAT). Verifying that a guessed poly-424 nomial f lies in the ideal induced by $G_{\rm sub}$ corresponds to showing that f equals zero for 425 all signals that can pass through the circuit. To show this, we assume $f \neq 0$ and show 426 unsatisfiability. We first encode each AIG gate that belongs to G_{sub} into conjunctive normal 427 form (CNF). To translate $f \neq 0$, we split it into two pseudo-Boolean (PB) constraints f < 0428 and f > 0, both of which we translate into CNF using PBLIB [22]. PBLIB does not offer a 429 direct encoding of \neq . We make two SAT calls using the SAT solver KISSAT [3]: one for the 430 constraint f < 0 and another for f > 0. If f = 0, both calls have to return "unsatisfiable". 431

We acknowledge that we could also use a PB solver, such as ROUNDINGSAT [7], directly without making the detour of translating everything to CNF. We have tried this, and there as no computational difference. All instances that could be solved using KISSAT could also be solved by ROUNDINGSAT in the same time. However, we decided to go with KISSAT because it was easier to integrate directly into our C++ implementation.

If one instance f < 0 or f > 0 is satisfiable, we collect the satisfying assignment provided by KISSAT as a witness for the wrong guess and add it as a sample in the next iteration of the guess-and-prove algorithm. This ensures that the incorrect guess f cannot reappear in future iterations, leading to a faster convergence of the guessing procedure.

Moreover, by using Lemma 15, we can also quickly identify situations when the considered subcircuit was chosen too small. We are only interested in finding a linear polynomial in G_{sub} that involves the variable lm(g). If, at some point in Algorithm 2, the candidate set Ldoes not contain any polynomial involving this variable, then Lemma 15 implies that no such polynomial can exist in G_{sub} (even if the candidate set L still contains incorrect guesses). Thus, in such a situation, we can immediately abort Algorithm 2 and extend G_{sub} .

Line 10-13: In case our linearization was successful, we extract p_{lin} from L to reduce the specification. If no polynomial p_{lin} was found after choosing G_{sub} as large as possible, we return *false*. We reduce the linearized specification S_{lin} by p_{lin} . After the main loop terminates, that is, when S_{lin} cannot be reduced any further, we return whether the final result is equal to zero.

¹ The algebraic closure $\overline{\mathbb{K}}$ of a field \mathbb{K} is the smallest algebraically closed field containing \mathbb{K} . Every field has an algebraic closure. For instance, $\overline{\mathbb{R}} = \mathbb{C}$.

452 4.1 Optimizations

453 Caching. To minimize algebraic computations, we cache the computed linear basis L for 454 subcircuits. Whenever we encounter an isomorphic subcircuit, i.e., one that is structurally 455 equivalent up to variable renaming, we reuse the stored linear basis L. To compare subcircuits 456 for this kind of equivalence, we map the variables of both circuits to a set of "standardized" 457 variables. Then we can compare the polynomial encodings of the circuits for syntactic 458 equality. In our application, we have observed cases where more than 99% of the utilized 459 subcircuits are cached.

Extract all linear polynomials. In line 10 of Algorithm 3 we actually do not only extract p_{lin} , but all linear polynomials of L. Since p_{lin} is the root of the subcircuit, all other linear polynomials in L are topologically smaller and may be required to reduce S_{lin} at later stages. *Propagating Vanishing Constraints*. In [12], only pairs of nodes with equal input nodes are marked as vanishin monomials. We extend this by further propagating the vanishing monomials along the positive parent nodes. For example, consider a detected vanishing monomial $g_1g_2 = 0$, and assume that there are gates with $g_3 - g_1(1 - g_4)$, $g_5 - g_2g_6$, where

467 g_1 and g_2 are positive inputs, then we also have $g_2g_3 = g_1g_5 = g_3g_5 = 0$.

468 **5** Experiments

⁴⁶⁹ We have implemented Algorithm 3 in a new C++ tool, called TALISMAN. We highlight that ⁴⁷⁰ TALISMAN is not restricted to circuit verification, but it can handle any AIG.

We now evaluate TALISMAN and compare it to related work [11–13, 16] using a total of 207 integer multiplier benchmarks. All benchmarks represent correct multipliers, meaning the circuits satisfy their specifications. We consider two types of benchmarks: *structured circuits*, where the components of a multiplier are clearly recognizable, and *synthesized circuits*, where gates are merged and rewritten to optimize the circuit, blurring component boundaries and complicating direct verification. We consider the following two sets:

1. Structured aoki-multipliers [9]: This set of benchmarks is generated by combining different 477 architectures for PPG, PPA, and FSA², yielding 192 structured multiplier architectures. 478 All of these circuits have an input bit-width of 64 and consist of 38 000 to 52 000 nodes. 479 2. Synthesized ABC-multipliers [1]: We generate a multiplier using ABC, consisting of a 480 simple PPG, an array PPA, and a ripple-carry FSA for bitwidths 32, 64, and 128. We 481 optimize the multipliers using four standard synthesis scripts (resyn, resyn2, resyn3, dc2) 482 and a complex script that combines multiple techniques³. These 15 optimized benchmarks 483 showcase the robustness of our approach. The node size ranges between 8000 and 130000. 484

⁴⁸⁵ Our experiments are conducted on a cluster of dual-socket AMD EPYC 7313 @ 3.7GHz
⁴⁸⁶ machines running Ubuntu 24.04. The time is listed in rounded seconds (wall-clock time).
⁴⁸⁷ We set the time limit to 300 s and the memory limit to 25 000 MB.

PPG: simple (sp), Booth encoding (bp); PPA: Array (ar), Wallace tree (wt), Balanced delay tree (bd), Overturned-stairs tree (os), Dadda tree (dt), (4;2) compressor tree (ct), (7,3) counter tree (cn), Red. binary addition tree (ba); FSA: Ripple-carry (rc), Carry look-ahead (cl), Ripple-block carry look-ahead (rb), Block carry look-ahead (bc), Ladner-Fischer (lf), Kogge-Stone (ks), Brent-Kung (bk), Han-Carlson (hc), Conditional sum (cn), Carry select (cs), Carry-skip fix size (csf), Carry-skip var. size (csv)

³ -c "logic; mfs2 -W 20; ps; mfs; st; ps; dc2 -l; ps; resub -l -K 16 -N 3 -w 100; ps; logic; mfs2 -W 20; ps; mfs; st; ps; iresyn -l; ps; resyn; ps; resyn2; ps; resyn3; ps; dc2 -l; ps;"



488 5.1 Results

In the following, we discuss the results of the experimental evaluation, splitting it into a
 comparison of TALISMAN with related work and an ablation study on TALISMAN.

We compare our tool TALISMAN to MULTILING [12], which also uses a linearization approach, as well as to DYNPHASEORDEROPT (DPOO) [16], AMULET2 [13], and TELUMA [11], all of which rely on lexicographic monomial orders. The results for the ABC benchmarks are shown in Figure 2 and for the aoki benchmarks in Figure 3. Tools using linearization approaches (MULTILING and our tool TALISMAN) are depicted in color. All tools using non-linear approaches are depicted in grayscale. We highlight the following key observations:

 While AMULET2 and TELUMA can solve all of the aoki benchmarks, their approaches are not robust under logic synthesis. AMULET2 solves only 4 ABC benchmarks, whereas TELUMA solves 13. In contrast, DPOO, MULTILING, and TALISMAN solve all of them.

DPOO solves 173 aoki benchmarks, while TALISMAN solves 136 benchmarks. However, among those 136 benchmarks are 11 that are not solved by DPOO.

⁵⁰² **3.** Among the 44 aoki benchmarks solved by MULTILING, 5 cannot be solved by TALISMAN⁴.

We examined these cases but could not identify any shared structural characteristics that

⁴ Those instances are: bp-ba-cl, bp-os-cl, sp-cn-cl, bp-cn-rc, and sp-cn-rc. All of them time out in the inner loop of Algorithm 3.

					TalisMan										
	F	Relat	ed Wo	rk	Total	otal Subcircuits FGLM Guess and						nd Prove			
Name	[13]	[11]	[16]	[12]	Time	#	%Ch	#	Time	#	Time	Guess	Eval	%Co	m.It.
abc64-cmp	TO	$\mathbf{E}\mathbf{E}$	1.1	7.3	1.9	4033	99.7	12	0.0	0	0	0	0	0	0
abc64- $rsn2$	TO	0.2	1.1	7.2	1.9	4033	99.7	12	0.0	0	0	0	0	0	0
abc128-cmp	TO	\mathbf{EE}	5.8	123.5	27.3	16257	99.9	12	0.0	0	0	0	0	0	0
abc128-rsn 2	TO	1.4	5.9	124.6	28.1	16257	99.9	12	0.0	0	0	0	0	0	0
bp-ar-ks	0.6	0.5	52.7	TO	205.9	6040	98.7	74	0.6	3	197.6	15087	3855	28.5	5
bp-ba-hc	0.8	0.6	-	TO	113.2	6147	97.6	144	0.6	1	104.2	4432	2754	48.0	3
bp-bd-rb	0.5	0.5	24.1	TO	75.6	5561	98.2	100	0.6	2	67.9	4452	1364	80.3	3
bp-cn-cl	10.2	2.5	-	155.8	124.4	2684	82.3	474	32.3	0	0	0	0	0	0
bp-ct-csv	0.9	0.3	10.2	TO	68.3	5204	88.2	614	40.4	1	17.3	864	864	100	1
bp-dt-lf	0.5	0.5	4.8	TO	94.0	5605	98.3	96	0.7	1	85.8	5091	2285	52.1	4
bp-os-rc	0.3	0.2	3.6	86.2	8.3	5697	98.1	108	0.7	0	0	0	0	0	0
bp-wt-csf	1.1	0.5	8.9	ТО	24.0	6007	98.0	117	0.7	1	14.8	800	800	100	1
sp-ar-rc	0.2	0.2	1.5	66.1	3.0	6509	99.7	18	0.0	0	0	0	0	0	0
sp-ba-csf	1.2	0.5	6.1	TO	26.3	8012	98.9	87	0.0	1	18.0	883	883	100	1
sp-bd-lf	0.5	0.4	2.8	TO	81.5	6403	99.4	40	0.0	1	76.3	5733	2077	52.4	5
sp-ct-cl	4.5	1.7	-	291.6	101.1	7782	95.7	335	25.4	0	0	0	0	0	0
sp-ct-hc	0.5	0.3	47.9	TO	98.2	7756	97.7	174	0.1	1	89.0	3675	2154	52.5	3
sp-dt-cs	0.4	0.3	2.5	TO	127.2	6958	99.2	57	0.0	2	120.5	5906	1735	83.7	3
sp-os-bc	0.4	0.4	9.5	MO	57.9	6420	99.2	50	0.0	2	52.7	6794	1137	84.9	4
sp-wt-ks	0.9	0.5	114.0	ТО	137.0	7639	99.3	49	0.0	1	129.0	4701	2861	48.1	3

Table 1 Statistics of selected benchmarks. TO: > 300 sec, MO: > 25000 MB, EE: error.

⁵⁰⁴ might explain MULTILING's advantage. We believe that the performance difference may ⁵⁰⁵ be due to variations in the heuristics used for subcircuit selection.

While we are still unable to fully outperform the *non-linear* lexicographic reasoning approaches, TALISMAN solves around three times more benchmarks and is an order of magnitude faster than the *linearization* approach of MULTILING. A direct comparison of the two linearization approaches is shown in Figure 4.

We present statistics on selected benchmarks in Table 1. The first block shows the timings 510 of related work, while the second block provides additional insights into our results. We 511 list the total time in rounded seconds. The subcircuits block lists the number of generated 512 subcircuits ("#") and the percentage of circuits found in the cache ("%Ch"). In FGLM, we 513 list the number of calls ("#") and the total time ("Time") spent in the FGLM algorithm. 514 In the Guess and Prove block, we also list the number of calls ("#") along with the total 515 time ("Time"). We further state the number of guessed polynomials ("Guess") and the 516 actual number evaluated ("Eval"), since we do not re-evaluate polynomials already found in 517 previous iterations. We show the percentage of correctly guessed polynomials ("%Co"), and 518 we also list the maximum number of iterations needed ("m.It."). A complete table, showing 519 statistics for all 207 benchmarks, is included in Appendix A. 520

Table 1 reveals some interesting facts: we can cache more than 80% of the circuits, even for synthesized ones. When there are fewer than 100 FGLM calls, computation time remains very low. In some cases, the guess-and-prove method immediately returns the correct polynomials. However, at most 5 iterations are needed to repair the incorrect guesses.

The time difference between "Total Time" and the "FGLM-time" and "Guess-and-Prove-Time" is mostly spent on the reduction of the specification. Only a negligible amount of computation time is spent on parsing and preprocessing. Around 30-40% of the time listed for "Guess and Prove" is spent in the SAT solver.

We also run an ablation study on TALISMAN. The results are summarized in Figure 5. We compare the default setting of TALISMAN to the following configurations: We only use



FGLM-style linearization (only-FGLM) or only guess-and-prove-style linearization (only-531 GaP) to compute linear polynomials. We turn off caching and always repeat all algebraic 532 computations (no-cache) or turn off identification and propagation of vanishing monomials 533 (no-van). We use repeated polynomial division (= ideal membership test) for verifying the 534 guessed polynomials, instead of using KISSAT (gap-uses-algred). And finally, since we also 535 use a different subcircuit heuristic, employ propagation of vanishing monomials, and use 536 caching, a direct comparison with MULTILING does not fully reflect the strengths of our 537 hybrid approach in comparison to using a degree-compatible Gröbner basis. Hence, we 538 have also added the option to extract linear polynomials by computing a degree-compatible 539 Gröbner basis using MSOLVE [2] in TALISMAN (use-GB). 540

It can be seen in Figure 5 that the default settings of TALISMAN and *no-cache* solve the 541 most benchmarks. Surprisingly, turning off caching does not have a huge negative impact 542 on the computation time. This indicates that caching a subcircuit is as expensive as simply 543 conducting the algebraic operations in our case. Turning off vanishing monomials leads to 544 a loss of 11 benchmarks. When forcing TALISMAN to use only one of our two proposed 545 approaches, we always lose benchmarks. Using only FGLM-style linearization leads to only 546 42 solved instances, while using only guess-and-prove-style linearization leads to 129 solved 547 benchmarks. We have included scatter plots in Figure 6 and Figure 7. While it can be seen in 548 Figure 6 that using only FGLM always worsens the situation, Figure 7 shows that there are 549 some benchmarks that benefit from using only the guess-and-prove approach. Using only the 550 guess-and-prove approach solved 18 benchmarks that are not solved in the default settings, 551 17 of which use a (7,3)-counter tree as PPA. Verifying guessed polynomials using algebra, or 552 using the Gröbner basis approach instead, lead to a tremendous loss of benchmarks. 553

554 **6** Conclusion

We have presented a hybrid approach for extracting linear polynomials from a set of polynomials. Our approach combines an FGLM-style linearization method with a guess-and-prove-style approach. We first present these algorithms at a general level before instantiating them for circuit verification. Our experimental evaluation shows that we outperform existing linearization techniques based on Gröbner bases. In the future, we aim to further improve our tool with more sophisticated algorithms for subcircuit detection. We also envision that our methods have applications beyond circuit verification, such as equivalence checking.

562		References
563	1	Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and
564		Verification, http://www.eecs.berkelev.edu/~alanmi/abc/, 2019. Bitbucket Version 1.01.
565	2	Jérémy Berthomieu, Christian Eder, and Mohab Safey El Din, msolve: A library for solving
566		polynomial systems. In Intl. Sumposium on Symbolic and Alaebraic Computation (ISSAC).
567		pages 51–58. ACM, 2021. doi:10.1145/3452143.3465545.
568	3	Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt.
569		CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Proc. of SAT
570		Competition 2024 – Solver. Benchmark and Proof Checker Descriptions, volume B-2024-1 of
571		Department of Computer Science Report Series B, pages 8–10. University of Helsinki, 2024.
572	4	Bruno Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes
573		nach einem nulldimensionalen Polynomideal. PhD thesis, University of Innsbruck, 1965.
574	5	David A. Cox, John Little, and Donal O'Shea. Ideals, varieties, and algorithms - an introduction
575		to computational algebraic geometry and commutative algebra (2. ed.). Undergraduate texts in
576		mathematics. Springer, 1997.
577	6	David A. Cox, John Little, and Donal O'Shea. Using Algebraic Geometry. Springer, 2005.
578	7	Jo Devriendt, Stephan Gocht, Emir Demirovic, Jakob Nordström, and Peter J. Stuckey.
579		Cutting to the core of pseudo-boolean optimization: Combining core-guided search with
580		cutting planes reasoning. In Conf. on Artificial Intelligence AAAI, pages 3750–3758. AAAI
581		Press, 2021. doi:10.1609/aaai.v35i5.16492.
582	8	Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. Efficient computation
583		of zero-dimensional gröbner bases by change of ordering. J. Symb. Comput., 16(4):329–344,
584		1993. doi:10.1006/jsco.1993.1051.
585	9	Naofumi Homma, Yuki Watanabe, Takafumi Aoki, and Tatsuo Higuchi. Formal design of
586		arithmetic circuits based on arithmetic description language. IEICE Trans. Fundam. Electron.
587		Commun. Comput. Sci., 89-A(12):3500-3509, 2006. doi:10.1093/ietfec/e89-a.12.3500.
588	10	Daniela Kaufmann. Formal Verification of Multiplier Circuits using Computer Algebra. PhD
589		thesis, Computer Science, Johannes Kepler University Linz, 2020.
590	11	Daniela Kaufmann, Paul Beame, Armin Biere, and Jakob Nordström. Adding dual variables
591		to algebraic reasoning for gate-level multiplier verification. In 2Design, Automation & Test in
592		Europe Conf. & Exhibition, DATE, pages 1431–1436. IEEE, 2022. doi:10.23919/DATE54114.
593		2022.9774587.
594	12	Daniela Kaufmann and Jérémy Berthomieu. Extracting linear relations from gröbner bases
595		for formal verification of and-inverter graphs. In Tools and Algorithms for the Construction
596		and Analysis of Systems TACAS, volume 15696 of LNCS, pages 355–374. Springer, Springer,
597		2025. doi:10.1007/978-3-031-90643-5_19.
598	13	Daniela Kaufmann and Armin Biere. Amulet 2.0 for verifying multiplier circuits. In Tools
599		and Algorithms for the Construction and Analysis of Systems TACAS, volume 12652 of LNCS,
600		pages 357-364. Springer, 2021. doi:10.1007/978-3-030-72013-1_19.
601	14	Daniela Kaufmann, Armin Biere, and Manuel Kauers. Verifying large multipliers by combining
602		SAT and computer algebra. In Formal Methods in Computer Aided Design, FMCAD, pages
603		28-36. IEEE, 2019. doi:10.23919/FMCAD.2019.8894250.
604	15	Daniela Kaufmann, Armin Biere, and Manuel Kauers. Incremental column-wise verification of
605		arithmetic circuits using computer algebra. Formal Methods Syst. Des., 56(1):22–54, 2020.
606		doi:10.1007/s10703-018-00329-2.
607	16	Alexander Konrad and Christoph Scholl. Symbolic computer algebra for multipliers revisited
608		- it's all about orders and phases. In Formal Methods in Computer-Aided Design FMCAD,
609		pages 261–271. IEEE, 2024. doi:10.34727/2024/isbn.978-3-85448-065-5_32.
610	17	Alexander Konrad, Christoph Scholl, Alireza Mahzoon, Daniel Große, and Rolf Drechsler.
611		Divider verification using symbolic computer algebra and delayed don't care optimization.
612		In Formal Methods in Computer-Aided Design FMCAD, pages 1–10. IEEE, 2022. doi:
613		10.34/2//2022/isbn.978-3-85448-053-2_17.

23:18 Guess and Prove

Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust boolean
 reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 21(12):1377–1394, 2002. doi:10.1109/TCAD.2002.804386.

⁶¹⁷ 19 Jinpeng Lv, Priyank Kalla, and Florian Enescu. Efficient gröbner basis reductions for formal

- verification of galois field arithmetic circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 32(9):1409–1420, 2013. doi:10.1109/TCAD.2013.2259540.
- Alireza Mahzoon, Daniel Große, and Rolf Drechsler. Polycleaner: clean your polynomials
 before backward rewriting to verify million-gate multipliers. In *Intl. Conf. on Computer-Aided Design ICCAD*, page 129. ACM, 2018. doi:10.1145/3240765.3240837.
- Alireza Mahzoon, Daniel Große, Christoph Scholl, and Rolf Drechsler. Towards formal
 verification of optimized and industrial multipliers. In *Design, Automation & Test in Europe Conf. & Exhibition, DATE*, pages 544–549. IEEE, 2020. doi:10.23919/DATE48585.2020.
 9116485.
- Tobias Philipp and Peter Steinke. Pblib A library for encoding pseudo-boolean constraints into CNF. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th Intl. Conf., Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *LNCS*, pages 9–16.
 Springer, 2015. doi:10.1007/978-3-319-24318-4_2.
- William A. Stein. Modular Forms: A Computational Approach, volume 79. American
 Mathematical Soc., 2007.
- ⁶³³ 24 Mertcan Temel. Vescmul: Verified implementation of s-c-rewriting for multiplier verification.
 ⁶³⁴ In Tools and Algorithms for the Construction and Analysis of Systems TACAS, volume 14570
- of LNCS, pages 340–349. Springer, 2024. doi:10.1007/978-3-031-57246-3_19.

A Statistics on Benchmarks

637 We present statistics for the full benchmark set used in our experimental evaluation.

Table 2 Statistics of TALISMAN and related work. TO: >	> 300 sec, MO: >	25000 MB, EE: error.
---	-------------------	----------------------

					TALISMAN										
	F	Relat	ed Wo	rk	Total	Subci	rcuits	FC	JLM		(Guess a	nd Pr	ove	
Name	[13]	[11]	[16]	[12]	Time	#	%Ch	#	Time	#	Time	Guess	Eval	%Co	m.It.
abc32-cmp	TO	$\mathbf{E}\mathbf{E}$	0.2	0.6	0.2	993	98.8	12	0.0	0	0	0	0	0	0
abc32-dc2	0.0	0.0	0.2	0.3	0.2	992	99.1	9	0.0	0	0	0	0	0	0
abc32-rsn	ТО	0.0	0.2	0.6	0.2	993	98.8	12	0.0	0	0	0	0	0	0
abc32-rsn2	ТО	0.0	0.3	0.6	0.2	993	98.8	12	0.0	0	0	0	0	0	0
abc32-rsn3	0.0	0.0	0.2	0.3	0.2	992	99.1	9	0.0	0	0	0	0	0	0
abc64-cmp	ТО	$\mathbf{E}\mathbf{E}$	1.1	7.3	1.9	4033	99.7	12	0.0	0	0	0	0	0	0
abc64-dc2	0.1	0.2	1.1	7.0	1.9	4032	99.8	9	0.0	0	0	0	0	0	0
abc64-rsn	ТО	0.2	1.0	7.5	1.9	4033	99.7	12	0.0	0	0	0	0	0	0
abc64-rsn2	ТО	0.2	1.1	7.2	1.9	4033	99.7	12	0.0	0	0	0	0	0	0
abc64-rsn3	0.1	0.2	1.1	7.2	1.9	4032	99.8	9	0.0	0	0	0	0	0	0
abc128-cmp	ТО	$\mathbf{E}\mathbf{E}$	5.8	123.5	27.3	16257	99.9	12	0.0	0	0	0	0	0	0
abc128-dc2	0.9	1.4	5.8	121.0	27.8	16256	99.9	9	0.0	0	0	0	0	0	0
abc128-rsn	ТО	1.4	5.8	116.5	28.3	16257	99.9	12	0.0	0	0	0	0	0	0
abc128-rsn2	ТО	1.4	5.9	124.6	28.1	16257	99.9	12	0.0	0	0	0	0	0	0
abc128-rsn3	0.9	1.4	5.8	128.3	27.4	16256	99.9	9	0.0	0	0	0	0	0	0
bp-ar-bc	0.4	0.5	11.9	ТО	33.1	6039	98.7	75	0.6	2	25.2	2253	848	87.3	2
bp-ar-bk	0.4	0.5	4.2	ТО	41.1	6042	98.7	74	0.6	3	33.2	4327	852	86.2	3
bp-ar-cl	2.6	1.1	243.2	ТО	30.6	2425	92.4	184	8.6	0	0	0	0	0	0
bp-ar-cn	0.5	0.4	6.1	ТО	ТО	-	-	-	-	-	-	-	-	-	-
bp-ar-cs	0.4	0.3	5.1	ТО	ТО	-	-	-	-	-	-	-	-	-	-
bp-ar-csf	0.4	0.5	9	ТО	17.2	6045	98.7	78	0.6	1	9.3	628	628	100	1
bp-ar-csv	0.5	0.5	6.2	ТО	18.4	6038	98.7	75	0.6	1	10.5	673	673	100	1
bp-ar-hc	0.5	0.5	11.0	ТО	51.9	6040	98.8	74	0.6	1	44.1	2807	1557	55.9	3
bp-ar-ks	0.6	0.5	52.7	ТО	205.9	6040	98.7	74	0.6	3	197.6	15087	3855	28.5	5
bp-ar-lf	0.4	0.5	4.6	ТО	51.4	6040	98.8	74	0.6	1	43.4	3589	1438	59.5	4
bp-ar-rb	0.4	0.5	10.9	ТО	52.5	6040	98.7	75	0.6	3	44.6	3813	1011	84.3	2
bp-ar-rc	0.3	0.2	3.5	70.7	7.8	6132	98.7	80	0.6	0	0	0	0	0	0
bp-ba-bc	0.5	0.5	12.2	ТО	58.3	6148	97.6	144	0.7	2	49.8	4042	1188	83.9	3
bp-ba-bk	0.5	0.5	14.3	ТО	83.1	6148	97.6	144	0.7	2	74.6	5890	1987	56.4	3
bp-ba-cl	5.7	2.4	TO	135.3	ТО	-	-	-	-	-	-	-	-	-	-
bp-ba-cn	0.5	0.4	46.1	MO	ТО	-	-	-	-	-	-	-	-	-	-
bp-ba-cs	0.4	0.3	5.6	ТО	TO	-	-	-	-	-	-	-	-	-	-
bp-ba-csf	1.3	0.5	8.2	ТО	27.8	6147	97.6	144	0.7	1	18.2	894	894	100	1
bp-ba-csv	1.2	0.5	8.5	ТО	28.0	6147	97.6	144	0.7	1	18.6	898	898	99.8	1
bp-ba-hc	0.8	0.6	TO	ТО	113.2	6147	97.6	144	0.6	1	104.2	4432	2754	48.0	3
bp-ba-ks	1.2	0.6	151.6	ТО	213.8	6147	97.6	144	0.7	1	204.4	7493	3969	41.5	4
bp-ba-lf	0.6	0.5	5.2	ТО	TO	-	-	-	-	-	-	-	-	-	-
bp-ba-rb	0.5	0.5	13.7	MO	85.6	6148	97.6	144	0.7	2	77.1	4728	1480	78.7	3
bp-ba-rc	0.3	0.3	3.6	81.2	8.5	6275	97.7	147	0.7	0	0	0	0	0	0
bp-bd-bc	0.4	0.5	10.0	ТО	41.9	5560	98.2	100	0.6	1	34.0	3831	1057	89.8	4
bp-bd-bk	0.5	0.5	5.0	ТО	51.6	5563	98.2	99	0.6	2	43.6	3801	1110	85.0	3
bp-bd-cl	4.8	2.0	TO	125.5	98.5	2305	87.8	280	27.8	0	0	0	0	0	0
bp-bd-cn	0.5	0.4	6.0	MO	TO	-	-	-	-	-	-	-	-	-	-
bp-bd-cs	0.4	0.3	5.7	ТО	TO	-	-	-	-	-	-	-	-	-	-
bp-bd-csf	1.3	0.5	8.5	ТО	23.9	5567	98.2	102	0.6	1	15.0	814	814	100	1
bp-bd-csv	0.9	0.5	8.6	ТО	26.0	5560	98.2	100	0.7	1	17.0	858	858	99.9	1
bp-bd-hc	0.6	0.5	28.9	ТО	85.0	5562	98.2	99	0.6	1	76.9	3666	2060	54.9	3
bp-bd-ks	0.8	0.6	128.6	TO	150.6	5560	98.2	99	0.6	1	142.2	4909	2841	50.6	3

						TAI	Tel	A A M								
	F	Polat	od Wo	rk		Subci	rouite	F(TAI TM	191	Guess and Prove					
Name	[13]	[11]	[16]	[12]	Total Time	#	%Ch	#	Time	-#	Time	Guess a	Eval	%Co	m It	
		[11]				"	7001					4.2.0.00		/000		
bp-bd-lf	0.5	0.5	4.9	EE	138.5	5564	98.2	99	0.6	3	130.6	12083	2725	41.1	4	
bp-bd-rb	0.5	0.5	24.1	ТО	75.6	5561	98.2	100	0.6	2	67.9	4452	1364	80.3	3	
bp-bd-rc	0.3	0.2	3.6	70.1	7.9	5681	98.2	101	0.7	0	0	0	0	0	0	
bp-cn-bc	5.4	0.7	267.8	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-bk	5.5	0.7	148.8	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-cl	10.2	2.5	ТО	155.8	124.4	2684	82.3	474	32.3	0	0	0	0	0	0	
bp-cn-cn	1.3	1.1	51.1	MO	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-cs	1.3	1.0	123.0	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-csf	5.8	0.6	ТО	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-csv	6.5	0.7	268.0	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-hc	5.7	0.7	ТО	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-ks	6.0	0.8	ТО	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-lf	5.5	0.7	152.3	ТО	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-rb	5.4	0.7	271.3	MO	ТО	-	-	-	-	-	-	-	-	-	-	
bp-cn-rc	5.3	0.7	ТО	87.1	ТО	-	-	-	-	-	-	-	-	-	-	
bp et be		03	37 /	то	102.6	5205	88.2	614	40.4	2	523	6732	11/1	83.8	1	
bp-ct-bc	0.4	0.0	18		06.2	5205	88.2	613	40.4	2	45.6	3837	1159	82.5	3	
bp-ct-bk	1 1	1.0	4.0 TO	252.2	1/3 1	4137	81 7	757	65.8	0	45.0	0007	1102	02.0		
bp-ct-cr	1.1 0 4	0.3	61	MO	т0	1101	01.1	101	00.0	0	0	0	0	0	0	
bp-ct-cn	0.4	0.0	5.4			-	-	-	-	-	-	-	-	-	-	
bp-ct-csf	1.4	0.0	8.6		66.0	5211	88.2	616	40.3	1	15 5	821	821	100	1	
bp-ct-csv	0.0	0.0	10.2	TO	68.3	5204	88.2	614	40.5	1	17.3	864	864	100	1	
bp-ct-bc	0.5	0.0	21.0	TO	132.0	5204	88.2	613	40.4	1	81.5	3703	2001	54.7	2	
bp-ct-ks	1.0	$0.0 \\ 0.4$	129.4	TO	200.9	5200	88.2	613	40.0	1	150.2	4978	2031	48 7	3	
bp-ct-lf	0.5	0.4	120.4	TO	126 7	5204	88.2	613	40.2	1	76.4	4910	2111	53.6		
bp-ct-rb	0.5	0.0	16.9	TO	110 /	5205	88.2	614	40.5	2	68.4	4051	1332	82.8	1 3	
bp-ct-rc	0.3	0.3 0.2	3.5	159.9	50.3	5325	88.4	616	40.0	0	00.4	1415	1002	02.0	0	
<u></u>		0.2		100.0	00.0	0020		1	10.1	-					-	
bp-dt-bc	0.5	0.5	14.4	ТО	74.6	5606	98.2	97	0.7	3	66.5	6975	1239	80.0	3	
bp-dt-bk	0.4	0.5	4.7	TO	72.8	5607	98.2	96	0.7	3	64.8	7317	1253	78.9	4	
bp-dt-cl	5.1	2.2	TO	142.9	125.7	2229	90.2	218	31.6	0	0	0	0	0	0	
bp-dt-cn	0.5	0.4	6.5	MO	TO	-	-	-	-	-	-	-	-	-	-	
bp-dt-cs	0.4	0.3	5.4	TO	TO	-	-	-	-	-	-	-	-	-	-	
bp-dt-csf	0.8	0.5	8.8	TO	24.8	5611	98.2	101	0.7	1	16.4	847	847	100		
bp-dt-csv	1.4	0.5	9.2	TO	27.4	5604	98.2	97	0.7	1	18.4	899	899	99.8		
bp-dt-nc	0.0	0.5	92.7	10 TO	107.4	5605	98.3	90		1	99.2	5068	2200	54.1 20.1	4	
bp-at-ks	1.0	0.0	110.2		201.0	5005	98.2	90		1	259.0	10504	3812	59.1	3	
bp-at-If	0.5	0.5	4.8		94.0	5605	98.3	90		1	85.8	5091	2285	52.1		
bp-at-rb	0.0	0.0	14.2	78.0	09.2	5005	98.2	97		2	51.2	5002	11/8	84.1		
bp-at-re	0.5	0.2	3.0	10.9	0.1	5750	96.2	100	0.7	0	0	0	0	0	0	
bp-os-bc	0.4	0.4	10.1	MO	40.2	5575	98.1	107	0.7	1	32.1	2897	1052	90.9	3	
bp-os-bk	0.4	0.4	4.6	ТО	53.3	5578	98.1	105	0.7	2	45.2	3838	1154	82.4	3	
bp-os-cl	5.4	1.9	ТО	133.6	ТО	-	-	-	-	-	-	-	-	-	-	
bp-os-cn	0.5	0.4	6.1	MO	TO	-	-	-	-	-	-	-	-	-	-	
bp-os-cs	0.5	0.4	5.5	ТО	TO	-	-	-	-	-	-	-	-	-	-	
bp-os-csf	1.4	0.4	8.6	ТО	24.9	5582	98.0	110	0.7	1	15.6	821	821	100	1	
bp-os-csv	1.0	0.4	8.3	ТО	26.0	5575	98.1	107	0.7	1	17.2	867	867	99.7	1	
bp-os-hc	0.6	0.4	23.5	TO	96.5	5577	98.1	105		1	88.2	3725	2175	52.5		
bp-os-ks	1.0	0.5	115.8	TO	169.5	5575	98.1	105	0.7	1	161.0	4978	2964	49.0	3	
bp-os-lf	0.5	0.4	5	ТО	83.8	5577	98.1	105	0.7	1	75.8	4809	2075	54.5	4	
bp-os-rb	0.5	0.4	23.3	'TO	77.0	5576	98.0	107		2	69.0	4471	1325	83.2		
bp-os-rc	0.3	0.2	3.6	86.2	8.3	5697	98.1	108	0.7	0	0	0	0	0	0	
bp-wt-bc	0.4	0.5	11.3	ТО	39.1	6000	98.1	115	0.7	1	30.9	2823	1035	90.0	3	
bp-wt-bk	0.5	0.5	4.9	ТО	40.5	6002	98.1	113	0.7	1	32.1	2796	1038	89.2	3	
bp-wt-cl	3.9	1.9	TO	135.1	89.3	2276	85.2	336	22.6	0	0	0	0	0	0	

					TALISMAN										
	F	Relat	ed Wo	rk	Total	Subci	rcuits	FC	GLM		(Juess a	nd Pr	ove	
Name	[13]	[11]	[16]	[12]	Time	#	%Ch	#	Time	#	Time	Guess	Eval	%Co	m.It.
bp-wt-cn	0.5	0.4	5.9	MO	TO	-	-	-	-	-	-	-	-	-	-
bp-wt-cs	0.4	0.3	5.3	ТО	ТО	-	-	-	-	-	-	-	-	-	-
bp-wt-csf	1.1	0.5	8.9	ТО	24.0	6007	98.0	117	0.7	1	14.8	800	800	100	1
bp-wt-csv	0.8	0.5	8.6	ТО	25.1	6000	98.1	115	0.7	1	16.3	847	847	99.5	1
bp-wt-hc	0.6	0.5	121.6	ТО	86.7	6002	98.1	113	0.7	1	77.9	3602	2087	53.2	3
bp-wt-ks	0.9	0.6	119.9	ТО	154.8	6000	98.1	113	0.7	1	145.8	4798	2779	50.8	3
bp-wt-lf	0.5	0.5	4.7	то	78.5	6002	98.1	113	0.7	1	70.0	4655	1950	56.4	4
bp-wt-rb	0.5	0.5	22.5	ТО	75.3	6001	98.0	115	0.7	2	66.7	4379	1337	80.5	3
bp-wt-rc	0.3	0.2	3.5	76.9	8.5	6118	98.1	116	0.7	0	0	0	0	0	0
sp-ar-bc	0.3	0.2	2.6		15.5	6448	99.8	14		$\frac{1}{2}$	12.6	3451	600	83.2	4
sp-ar-bk	0.3	0.2	1.0	TO	11.2	6448	99.8	15	0.0	1	8.3	1483	540	90.9	3
sp-ar-cl	0.0	0.2	27.8	74.0	5.4	3999	98.1	75	1.2		0.0	1100	010	00.0	
sp-ar-cn	0.0	0.3	2.6	то	55.9	6459	99.4	37	0.2	$\frac{1}{2}$	52 5	4077	1197	83 1	4
sp-ar-cs	0.1	0.3	2.0	TO	29.8	6460	99.4	37	0.2	3	26.8	1441	732	98.8	1
sp-ar-csf	0.1	0.0	2.1		67	6451	99.7	16	0.2	1	3.8	425	425	100	1
sp-ar-csv	0.3	0.2	2.5	EE	74	6447	99.8	14	0.0	1	4.5	455	455	100	1
sp-ar-hc	0.0	0.2	1.0		22.0	6448	99.0	15	0.0	1	10.0	1781	967	58.2	3
sp-ar-ks	0.3	0.2	15.8		36.8	6447	99.8	14	0.0	1	32.5	2303	1282	54.5	3
sp-ar-lf	0.3	0.2	22	TO	21.2	6448	99.8	15	0.0	1	17.6	2344	879	64.0	
sp-ar-rh	0.3	0.2	2.2 2.4		16.6	6447	99.8	14	0.0	1	13.5	1737	639	89.7	3
sp-ar-rc	$0.0 \\ 0.2$	0.2	1.5	66.1	3.0	6509	99.7	18	0.0		10.0	0	000	0	0
sp-ba-bc	0.5	0.5			55 7	8013	08.0	87		2	185	3008	1172	8/1	<u> </u>
sp-ba-be	0.0	0.5	11.4		- 00.1 ТО	0010	50.5	01	0.0	2	40.0	0000	1112	04.1	5
sp-ba-cl	5.2	2.0	т. ТО	EE I										_	
sp-ba-cn	0.2	0.4	23		TO	_	_	_	_	_	_	_	_	_	_
sp-ba-cs	0.4	0.4	2.6											_	
sp-ba-csf	1.2	0.5	2.0 6.1		26.3	8012	98.9	87	0.0	1	18.0	883	883	100	1
sp-ba-csi	1.2	0.5	81		20.5	8012	08.0	88	0.0	1	18.1	887	887	00.8	1
sp-ba-csv	0.8	0.5	158.6		20.5 TO	0012	50.5	00	0.0	1	10.1	001	001	55.0	1
sp-ba-lic	1.3	0.0	130.0 137 0			-	-	-	-	-	-	-	-	-	-
sp-ba-ks	1.0	0.0	2 1		73	8240	083	1/2	0.1		-	-		-	0
sp-ba-rh	0.7	0.5	10.1		т.5 ТО	0249	50.5	142	0.1		0			0	0
sp-ba-rc	0.0	0.0	1 7	EE EE	73	81/0	98.9	03						0	
sp bd be	0.1	0.2			3/ 8	6402	00.4					0		01.8	<u> 2</u>
sp-bu-bu	0.5	0.4	2.0		187	6404	99.4	40		1	44.9	5808	1010	91.0 91.1	
sp-bd-bk	11	1.7	2.0 TO	105 7	85.5	4119	99.5	182	22.5		44.2	0030	1035	04.1	0
sp-bd-cr	4.4	1.1	25	105.7 MO	00.0 TO	4112	95.0	102	22.0	0	0	0		0	0
sp-bd-cn	0.5	0.4	0.0 9.5			-	-	-	-	-	-	-	-	-	-
sp-bu-cs	0.4	0.3	2.5 5.0		10.6	6408	00.3	13	0.0	1	14.4	702	702	100	1
sp-bd-csi	1.1	0.4	6.5		19.0 91.6	6400	00.4	40		1	16.9	838	838	00.0	1
sp-bd-csv	1.1	0.4	25.3		70.0	6402	00.4	40		1	74.3	3506	2027	54.4	2
sp-bu-lic	1.0	0.4	20.0		19.9	6403	99.4	20			128 0	4789	2021	51 /	2
sp-bd-ks	1.2	0.5	90.1		104.4 01 5	6402	99.4	10			76.2	5799	2124	59.4	5
sp-bd-n	0.5	0.4	2.0		72.0	6403	99.4	40	0.0	1	69 5	5495	12/01	70.2	
sp-bd-rc	0.5	$0.4 \\ 0.2$	0.0 1.6	69.8	13.0 4.6	6520	99.3 99.4	$\frac{40}{42}$	0.0		00.5	0435	1349	19.5	4
sp bu ie	0.0	0.2	1.0			0020	00.1	12	0.0					0	
sp-cn-bc	3.2	0.7	51.0	TO	TO	-	-	-	-	-	-	-	-	-	-
sp-cn-bk	3.2	0.1	28.2	164.0		-	-	-	-	-	-	-	-	-	-
sp-cn-ci	1.0	2.3 1.0	11.0	104.0		-	-	-	-	-	-	-	-	-	-
sp-cn-cn	1.0	1.2	11.2			-	-	-	-	-	-	-	-	-	-
sp-cn-cs	1.5	1.2	10.4			-	-	-	-	-	-	-	-	-	-
sp-cn-cst	4.0	0.7	13.4			-	-	-	-	-	-	-	-	-	-
sp-cn-csv	4.3	0.7	14			-	-	-	-	-	-	-	-	-	-
sp-cn-hc	3.3	0.7	104.2			-	-	-	-	-	-	-	-	-	-
sp-cn-ks	3.7	0.8	124.9	OT	TO	-	-	-	-	-	-	-	-	-	-

	1				TalisMan										
	F	Relat	ed Wo	rk	Total	Subci	rcuits	FC	ΞLM		(Guess a	nd Pr	ove	
Name	[13]	[11]	[16]	[12]	Time	#	%Ch	#	Time	#	Time	Guess	Eval	%Co	m.It.
sp-cn-lf	3.2	0.7	24.1	TO	TO	-	-	-	-	-	-	-	-	-	-
sp-cn-rb	3.2	0.7	23.9	MO	ТО	-	-	-	-	-	-	-	-	-	-
sp-cn-rc	3.0	0.7	177.8	139.3	TO	-	-	-	-	-	-	-	-	-	-
sp-ct-bc	0.3	0.3	10.0	TO	71.5	7757	97.7	174	0.1	3	62.6	8277	1175	80.9	4
sp-ct-bk	0.3	0.3	2.1	ТО	53.0	7757	97.7	174	0.1	2	44.1	3800	1133	83.2	3
sp-ct-cl	4.5	1.7	ТО	291.6	101.1	7782	95.7	335	25.4	0	0	0	0	0	0
sp-ct-cn	0.3	0.3	2.8	MO	ТО	-	-	-	-	-	-	-	-	-	-
sp-ct-cs	0.3	0.2	2.5	ТО	ТО	-	-	-	-	-	-	-	-	-	-
sp-ct-csf	1.2	0.2	6.2	ТО	25.2	7761	97.7	176	0.1	1	15.0	814	814	100	1
sp-ct-csv	1.2	0.3	7.4	TO	26.7	7755	97.7	174	0.1	1	16.7	860	860	99.8	1
sp-ct-hc	0.5	0.3	47.9	TO	98.2	7756	97.7	174	0.1	1	89.0	3675	2154	52.5	3
sp-ct-ks	0.7	0.3	108.7	ТО	163.2	7755	97.8	173	0.1	1	153.6	4928	2915	49.4	3
sp-ct-lf	0.4	0.3	2.9	TO	90.3	7756	97.7	174	0.1	1	81.2	5942	2211	50.7	5
sp-ct-rb	0.4	0.3	9.3	TO	77.7	7756	97.7	174	0.1	2	68.3	4464	1362	80.6	3
sp-ct-rc	0.2	0.2	1.6	269.1	9.1	7875	97.7	180	0.1	0	0	0	0	0	0
sp-dt-bc	0.5	0.5	5.4	TO	71.5	6959	99.1	57	0.0	3	64.7	6717	1296	76.8	3
sp-dt-bk	0.5	0.4	2.1	TO	68.1	6960	99.1	58	0.0	3	61.3	7084	1258	78.1	4
sp-dt-cl	5.5	2.1	ТО	109.6	121.9	3971	95.6	175	30.3	0	0	0	0	0	0
sp-dt-cn	0.4	0.4	3.8	MO	ТО	-	-	-	-	-	-	-	-	-	-
sp-dt-cs	0.4	0.3	2.5	TO	127.2	6958	99.2	57	0.0	2	120.5	5906	1735	83.7	3
sp-dt-csf	1.9	0.4	6.7	TO	24.9	6963	99.1	61	0.0	1	16.4	850	850	100	1
sp-dt-csv	2.1	0.4	6.5	EE	27.2	6957	99.2	57	0.0	1	18.6	898	898	99.7	
sp-dt-hc	0.6	0.5	27.3	TO	99.7	6958	99.2	58	0.0	1	92.7	3855	2213	53.5	
sp-at-ks	1.1	0.5	149.1		1/1.4	6957 COTO	99.2	57			164.1	5187	3039	49.5	3
sp-dt-lf	0.5	0.4	2.8		89.2	6958 6050	99.2	08 57		1	82.2	0591	1521	02.0 75 0	
sp-dt-ro	0.5 0.4	0.4	12.5	01.8	7.0	0959	99.1	- 57 - 58		0	101.1	9581	1001	10.2	
sp og be	0.1	0.2	0.5		57.0	6420	00.2	50		0 0	59.7	6704	1127	84.0	
sp-os-bk	0.4	0.4	9.0		64.4	6420	99.2 00.2	50		2	59.2	6945	12/3	77 1	4
sp-os-cl	45	1.0	2.1 TO	00 6	106.2	4102	99.2 95.0	206	27.3		03.2	0340	1240	11.1	1
sp-os-cr	0.5	$1.5 \\ 0.4$	36	MO	TO	-1102		200	21.5	-	-	-	-	-	-
sp-os-cs	0.4	0.3	2.5	TO	TO	-	_	_	-	_	_	-	_	_	_
sp-os-csf	1.6	0.3	5.8	TO	21.9	6425	99.2	53	0.0	1	15.4	828	828	100	1
sp-os-csv	1.6	0.3	6.4	ТО	23.8	6419	99.2	50	0.0	1	17.3	874	874	99.9	1
sp-os-hc	0.6	0.4	34.4	ТО	122.7	6421	99.2	50	0.0	2	116.3	7582	2587	44.5	3
sp-os-ks	0.9	0.4	105.7	ТО	242.6	6420	99.2	49	0.0	2	236.1	11547	3766	38.9	4
sp-os-lf	0.5	0.4	2.8	ТО	82.6	6420	99.2	50	0.0	1	77.2	4879	2156	53.0	4
sp-os-rb	0.5	0.3	8.9	TO	75.0	6420	99.2	50	0.0	2	69.8	4520	1345	82.9	3
sp-os-rc	0.3	0.2	1.6	90.1	5.2	6542	99.2	52	0.0	0	0	0	0	0	0
sp-wt-bc	0.5	0.4	7.1	ТО	39.2	7639	99.3	51	0.0	1	31.6	3682	1011	90.3	4
$\operatorname{sp-wt-bk}$	0.5	0.4	2.1	ТО	37.5	7640	99.3	50	0.0	1	30.0	2733	1029	87.8	3
sp-wt-cl	4.4	1.7	ТО	116.8	79.1	4064	93.4	269	20.1	0	0	0	0	0	0
sp-wt-cn	0.5	0.3	3.6	MO	TO	-	-	-	-	-	-	-	-	-	-
sp-wt-cs	0.4	0.3	2.5	TO	TO	-	-	-	-	-	-	-	-	-	-
sp-wt-cst	1.1	0.4		TO	22.1	7645	99.3	53		1	13.8	778	778	100	
sp-wt-csv		0.4	5.8	TO	23.6	7639	99.3	51			15.4	823	823	99.9	
sp-wt-nc	0.7	0.5	30.1		93.0	7620	99.3	00 20			00.8	3514	2038	03.1 49 1	J 9
sp-wt-Ks	0.9	0.0	114.0 9.0		131.0	7640	99.3 00.9	49 50			149.0 70 5	4701 5506	1062	40.1 51 /	3 5
sp-wt-n	0.0	0.4	2.9		57 2	7630	<i>୭୭</i> .୦ ପ୍ର ୧	51		1	40.5	4266	1202	87 /	1
sp-wt-rc	0.4	0.4 0.3	1.6	86.9	7.6	7753	99.3	54	0.0	0	19.7	1200 N	1207	01.4	
. r	2.1		1 1.0	0.0						ΙŬ	I 1	, J		5	I ~