A SAT Solver + Computer Algebra Attack on the Minimum Kochen–Specker Problem

Zhengyu Li[1] , **Curtis Bright**[2] , **Vijay Ganesh**[1]
[1]School of Computer Science, Georgia Institute of Technology, United States
[2]School of Computer Science, University of Windsor, Canada
brian.li@gatech.edu, cbright@uwindsor.ca, vganesh45@gatech.edu

Abstract

One of the fundamental results in quantum foundations is the Kochen–Specker (KS) theorem, which states that any theory whose predictions agree with quantum mechanics must be *contextual*, i.e., a quantum observation cannot be understood as revealing a pre-existing value. The theorem hinges on the existence of a mathematical object called a KS vector system. While many KS vector systems are known, the problem of finding the minimum KS vector system in three dimensions (3D) has remained stubbornly open for over 55 years.

To address the minimum KS problem, we present a new verifiable proof-producing method based on a combination of a Boolean satisfiability (SAT) solver and a computer algebra system (CAS) that uses an isomorph-free orderly generation technique that is very effective in pruning away large parts of the search space. Our method shows that a KS system in 3D must contain at least 24 vectors. We show that our sequential and parallel Cube-and-Conquer (CnC) SAT+CAS methods are significantly faster than SAT-only, CAS-only, and a prior CAS-based method of Uijlen and Westerbaan. Further, while our parallel pipeline is somewhat slower than the parallel CnC version of the recently introduced Satisfiability Modulo Theories (SMS) method, this is in part due to the overhead of proof generation. Finally, we provide the first computer-verifiable proof certificate of a lower bound to the KS problem with a size of 40.3 TiB in order 23.

1 Introduction

Quantum Mechanics (QM) is often described as one of the most successful physical theories of all time, and yet many questions regarding the foundations of QM continue to be hotly contested. Many interpretations of QM, i.e., mappings from mathematical formalisms of QM to physical phenomena, have been proposed in order to resolve these quantum foundational questions. Hidden-variable theories are attempts at understanding counterintuitive QM phenomena through a deterministic lens by positing the existence of unobservable entities or hidden variables [Held, 2000] that the standard QM theory does not account for.

The Kochen–Specker (KS) theorem, a fundamental result in quantum foundations, rules out non-contextual hidden-variable theories by establishing the existence of a finite set of three-dimensional (3D) vectors (a KS system) witnessing a contradiction between non-contextuality (i.e., the assumption that observables can be assigned values prior to measurement and independent of measurement context) and the SPIN axiom of QM. KS systems are also a crucial ingredient in the proof of the "Free Will Theorem" of [Conway and Kochen, 2006]. The very first KS vector system, discovered in 1967, contains 117 vectors [Kochen and Specker, 1967].

Since the publication of Kochen and Specker's theorem in 1967, physicists and mathematicians have wondered how many vectors the smallest-sized KS vector system contains (see Table 1 and Section 3). Finding the minimum KS system (the KS problem) is not only of scientific and historical interest, but also has applications in quantum information processing [Cañas *et al.*, 2014].

1.1 The SAT+CAS Paradigm for Hard Combinatorial Problems

Despite the impressive achievements of SAT solvers [Ganesh and Vardi, 2020], they struggle on certain problems such as those containing many symmetries [Metin *et al.*, 2018] or those requiring the usage of more advanced mathematical theories than propositional logic [Bright *et al.*, 2022]. Much work has been done to remedy these drawbacks, including the development of sophisticated symmetry breaking techniques [Aloul *et al.*, 2003] and the development of solvers that support richer logic such as "SAT modulo theories" or SMT solvers [Barrett *et al.*, 2016]. However, the mathematical support of SMT solvers is more focused on program analysis and verification. By contrast, computer algebra systems (CAS) provide rich mathematical functionality and are vast storehouses of mathematical knowledge that are particularly suitable for hard combinatorial problems that come from varied areas of math, physics, and computer science.

In response to this need for combining the efficient search capabilities of SAT solvers with the mathematical knowledge available in computer algebra systems, a new kind of "SAT+CAS" solving methodology was developed by [Zulkoski *et al.*, 2015; Bright *et al.*, 2016] and inde-

Discoverers	Year	Bound
Kochen and Specker	1967	≤ 117
Jost	1976	≤ 109
Conway and Kochen	1990	≤ 31
Arends *et al.*	2009	≥ 18
Uijlen and Westerbaan	2016	≥ 22
Li *et al.*	2022	≥ 23
Kirchweger *et al.*	2023	≥ 24

Table 1: A chronology of the bounds on the size of the minimum KS vector system in three dimensions.

pendently by [Ábrahám, 2015]. This SAT+CAS solving methodology has been successfully applied to many diverse problems, including verification [Kaufmann and Biere, 2023; Mahzoon *et al.*, 2022], computing directed Ramsey numbers [Neiman *et al.*, 2022], and finding Williamson matrices [Bright *et al.*, 2020b].

1.2 Isomorph-free Exhaustive Orderly Generation

In this paper, we use the SAT+CAS solving methodology to dramatically improve the performance of searching for KS systems compared to a SAT-only, CAS-only (based on the program *nauty* [McKay and Piperno, 2014]), and many previous approaches developed to prove lower bounds to the minimum KS problem (Section 1.3). This is made possible by combining the powerful search algorithms used in modern SAT solvers with an "isomorph-free exhaustive generation" approach, implemented via CAS, preventing duplicate exploration of isomorphic parts of the search space. This approach was recently used to resolve Lam's problem from projective geometry [Bright *et al.*, 2021]. Although isomorph-free exhaustive generation has been used extensively in combinatorial enumeration, it has only recently been combined with SAT solving [Junttila *et al.*, 2020; Savela *et al.*, 2020].

The traditional approach to prevent a SAT solver from repeatedly exploring isomorphic parts of a search space is via the use of *symmetry breaking* techniques [Metin *et al.*, 2018]. One such symmetry breaking approach is to add "static" constraints prior to the search that reduces the size of the search space [Crawford *et al.*, 1996; Heule, 2019]. Another approach is to "dynamically" break symmetries during the search [Sellmann and Hentenryck, 2005; Metin *et al.*, 2018] such as in the SAT Modulo Symmetries (SMS) paradigm [Kirchweger and Szeider, 2021]. Our approach is related in that it also dynamically adds constraints to the problem during the solving process.

The key difference between these previous approaches and ours is that we query a CAS in the inner loop of a SAT solver, and this permits learning more general constraints. For example, we call a CAS to detect the presence of certain subgraphs that are known to not be contained in any KS system; if such a subgraph is detected, it is dynamically blocked. Most importantly, our method produces independently verifiable proofs in a simple extension of the DRAT format [Wetzler *et al.*, 2014].

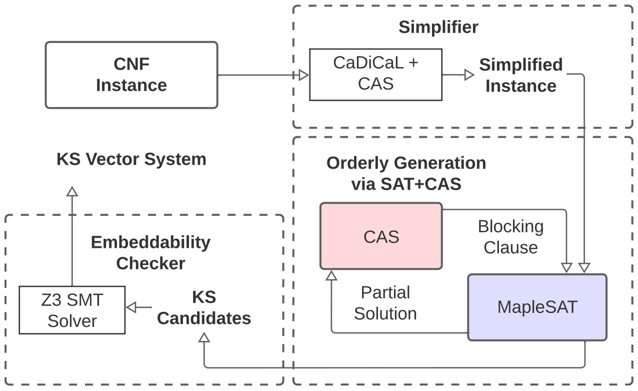

Figure 1: A flowchart of our SAT+CAS tool MATHCHECK for solving the KS problem in the sequential setting. The CNF instance encoding the KS problem (see Section 4) is simplified using CaDiCaL+CAS. The simplified instance is passed to the MapleSAT+CAS tool (see Section 5) either sequentially or in parallel using Cube-and-Conquer (CnC). Finally, an embeddability checker applies the SMT solver Z3 to determine whether the candidates are embeddable.

1.3 Our Contributions

Proof-producing SAT+CAS with Orderly Generation: In this paper, we present the design and implementation of a verifiable proof-producing SAT+CAS system with orderly generation (OG) aimed at combinatorial problems (as part of the SAT+CAS tool, MATHCHECK[1]), thus showing that the minimum KS system must contain at least 24 vectors. Also, we extend our work to complex vectors in three dimensions (3D), and thus establish a lower bound of 24 for both the real and complex KS problem.

Speedup over Competing Methods: We show that our sequential and parallel Cube-and-Conquer (CnC) SAT+CAS methods are significantly faster than SAT-only and CAS-based methods [Uijlen and Westerbaan, 2016]. Further, while our pipeline is somewhat slower than the recently introduced SMS method [Kirchweger *et al.*, 2023], this is in part due to the generation of verifiable proofs by our methods which added additional complexity and slowed down MATHCHECK.

Formal Verification of Results: Finally, our approach provides a formal verification of the lower bound of 24 for the minimum KS problem in 3D by verifying all certificates computed by MATHCHECK in all orders up to and including order 23 (see Section 9). By contrast, Kirchweger *et al.* describe a method to verify their result, but they only report having verified 5% of the proofs in order 23.

2 Background

We introduce several fundamental concepts here from quantum foundations such as the SPIN axiom, 010-colorability, the KS theorem, and the KS vector system. For a deeper dive, we refer the reader to the Stanford Encyclopedia of Philosophy [Held, 2000]. Regarding SAT solvers, we refer the reader to the Handbook of Satisfiability [Biere *et al.*, 2021].

[1]Code at https://github.com/BrianLi009/MathCheck

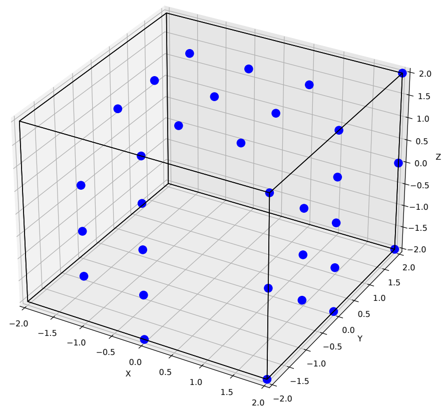

Figure 2: The 31 vectors of the smallest known KS system in 3D (discovered by Conway and Kochen circa 1990). For simplicity, the vectors have been scaled to lie on the cube with vertices $(\pm 2, \pm 2, \pm 2)$ instead of the unit sphere.

2.1 The KS Theorem

Informally, the KS theorem states that there is a contradiction between the SPIN axiom of standard QM and the assumption of non-contextuality. The Stanford Encyclopedia of Philosophy provides a comprehensive background to the KS theorem and stresses its importance in the foundations of QM [Held, 2000]. The proof of the KS theorem crucially relies on the existence of a KS vector system (see Figure 2). More precisely, exhibiting the existence of a KS vector system proves the KS theorem, which essentially states that the unit sphere is not 010-colorable (defined below).

Spin of an Elementary Particle: Spin is an intrinsic form of angular momentum of elementary particles whose existence can be inferred from the Stern–Gerlach experiment [Gerlach and Stern, 1922]. In our context, a spin-1 particle is shot through a magnetic field in a given direction and it will continue undisturbed, deflect up, or deflect down—corresponding to the three possible angular momentum states 0, 1, and -1. Thus, the square of this measurement is 0 or 1.

SPIN axiom: The SPIN axiom of QM states that the squared spin measurements of a spin-1 particle are 1, 0, 1 in any three pairwise orthogonal directions of measurement. Thus, the observable corresponding to the question "is the squared spin 0?" measured in three mutually orthogonal direction always produces *yes* in exactly one direction and *no* in the other two orthogonal directions. We use the dual of the above form in the present work, i.e., the '010' convention rather than '101', following Uijlen and Westerbaan. The SPIN axiom follows from the postulates of QM and is experimentally verifiable [Huang *et al.*, 2003].

KS Vector System: A KS vector system can be represented in multiple ways and we describe it as a finite set of points on a sphere. As a consequence of the SPIN axiom, the squared-spin measurements along opposite directions must yield the same outcome. Therefore, two collinear vectors are considered to be equivalent. To define a KS vector system, we first formally define a vector system and the notion of 010-colorability. For the purposes of this paper, we limit ourselves to the 3D version of the KS problem as the size of the minimum KS system in higher dimensions is already known [Pavičić *et al.*, 2005].

Definition 1 (Vector System). *A **vector system** is a finite set of non-collinear points on the unit sphere in* $\mathbf{R}^3$.

A $\{0,1\}$-coloring of a vector system is an assignment of 0 and 1 to each vector in the system. The colorings of interest to us are described in the following definition.

Definition 2 (010-Colorability of Vector Systems). *A vector system is **010-colorable** if there exists an assignment of 0 and 1 to each point such that:*

1. *No two orthogonal points are assigned 1.*
2. *Three mutually orthogonal points are not all assigned 0.*

Definition 3 (KS Vector System). *A **KS vector system** is one that is not 010-colorable.*

Definition 4 (Orthogonality Graph). *For a vector system* $\mathcal{K}$, *define its **orthogonality graph*** $G_{\mathcal{K}} = (V, E)$, *where* $V = \mathcal{K}$, $E = \{\, (v_1, v_2) : v_1, v_2 \in \mathcal{K}$ *and* $v_1 \cdot v_2 = 0 \,\}$.

Essentially, the vertices of $G_{\mathcal{K}}$ are the vectors in $\mathcal{K}$, and there is an edge between two vertices exactly when their corresponding vectors are orthogonal. Similarly, the notion of 010-colorability can be translated from a vector system to an orthogonality graph.

Definition 5 (010-colorability of Graphs). *A graph G is **010-colorable** if there is a* $\{0,1\}$*-coloring of the vertices such that the following conditions are satisfied simultaneously:*

1. *No two adjacent vertices are colored 1.*
2. *For each triangle, the vertices are not all colored 0.*

It is not always the case that an arbitrary graph has a corresponding vector system, but if one does exist then we say that such a graph is *embeddable*.

Definition 6 (Embeddable Graph). *A graph* $G = (V, E)$ *is **embeddable** if it is a subgraph of an orthogonality graph for some vector system.*

The embeddability of a graph G implies the existence of a vector system $\mathcal{K}$ whose vectors have a one-to-one correspondence with the vertices of G, such that adjacent vertices are assigned to orthogonal vectors. An example of an unembeddable graph is the cyclic graph C_4 on 4 vertices, as the orthogonality constraints force a pair of opposite vertices in C_4 to be mapped to collinear vectors (which are not allowed in a vector system).

Definition 7 (KS Graph). *An embeddable and non-010-colorable graph is called a **KS graph**.*

Observation 1. *There exists a KS vector system if and only if there exists a KS graph.*

In our work, we search for KS graphs using MathCheck, but because it is nontrivial to enforce the embeddability of a graph, we do not require solutions produced by MathCheck to be embeddable. Instead, we add a post-processing step ensuring solutions found by MathCheck (known as ***KS candidates***) are in fact embeddable (see Figure 1). However, MATHCHECK does ensure that KS candidates will not contain any unembeddable subgraphs of order less than or equal to 12 (see Section 6).

2.2 The Minimum KS Problem

The minimum KS problem is to find a KS vector system of minimum cardinality, that is, a system with the fewest number of vectors in 3D space (or equivalently a KS graph with the fewest number of vertices). Every KS system has an associated KS graph, so if a KS graph with cardinality n does not exist then a lower bound on the minimum KS problem is at least $n + 1$.

3 Previous Work

Over the last 55+ years, many mathematicians and physicists such as Roger Penrose, Asher Peres, and John Conway have attempted to find a minimum 3D KS system (see Table 1). The first KS system was constructed in 1967 and it contained 117 vectors [Kochen and Specker, 1967]. A KS system with 109 vectors was found in 1976 [Jost, 1976]. Peres found a KS system of size 33 in 1991 [Peres, 1991], and Schütte found a KS system of size 33 in 1996.

The current smallest known KS system in 3D contains 31 vectors and was discovered by Conway and Kochen circa 1990 [Conway and Kochen, 2002]. All these discoveries were made analytically, without the use of computational methods. Recently, an automated generation approach to robustly generate KS systems in odd dimensions was introduced [Pavičić and Megill, 2022], which led to the discovery of many more 3D KS systems.

In 2011, Arends *et al.* proved several interesting properties of KS graphs and leveraged them to computationally establish that a KS system must contain at least 18 vectors [Arends *et al.*, 2011]. Seven years later, Uijlen and Westerbaan showed that a KS system must have at least 22 vectors [Uijlen and Westerbaan, 2016]. This computational effort used around 300 CPU cores for three months and relied on the *nauty* software package [McKay and Piperno, 2014] to exhaustively search for KS graphs.

At the *2022 SC-Square workshop*, Li *et al.* presented searches for KS systems with up to 22 vectors. The searches were exhaustive and no KS systems were found, thus establishing for the first time a lower bound on the minimum KS problem of at least 23 vectors.

Following Li *et al.*, Kirchweger, Peitl, and Szeider [Kirchweger *et al.*, 2023] also completed a search for KS systems and established a lower bound of 24 vectors, but using an SMS solver and an alternate definition of canonicity. They do not use OG, as their definition of canonicity does not satisfy property (2) from Sec. 5, but otherwise the SMS approach is similar in that it combines a SAT solver with a canonical checking routine. Their approach can also be used to generate proof certificates, though they only report verifying 5% of the certificates in the order 23 search and do not report verifying the clauses dynamically learned by SMS. By contrast, our approach formally verifies all results without trusting the SAT solver or CAS.

4 SAT Encoding of the Minimum KS Problem

As stated earlier, every KS vector system $\mathcal{K}$ can be converted into a KS graph $G_{\mathcal{K}}$. Each vector in $\mathcal{K}$ is assigned to a vertex in $G_{\mathcal{K}}$, so that if two vectors are orthogonal, then their corresponding vertices are connected. We say a KS graph is minimal if the only subgraph that is also a KS graph is itself. In 2011, Arends *et al.* proved that a 3D minimal KS graph must satisfy the following properties:

1. The graph does not contain a square subgraph (C_4).
2. Each vertex of the graph has a minimum degree 3.
3. Each vertex of the graph is part of a triangle (C_3).

Below we describe encoding these properties and the non-010-colorability of KS graphs in conjunctive normal form (CNF). If a SAT solver produces solutions for such an encoding, then these solutions correspond to graphs satisfying all of the above-mentioned constraints.

A simple undirected graph of order n has $\binom{n}{2}$ potential edges, and we represent each edge as a Boolean variable. The edge variable e_{ij} is true exactly when the vertices i and j are connected, where $1 \leq i < j \leq n$. For convenience, we let both e_{ij} and e_{ji} denote the same variable since the graphs we consider are undirected. We also use the $\binom{n}{3}$ triangle variables t_{ijk} denoting that distinct vertices i, j, and k are mutually connected. In Boolean logic this is expressed as $t_{ijk} \leftrightarrow (e_{ij} \wedge e_{ik} \wedge e_{jk})$ which in conjunctive normal form is expressed via the four clauses $\neg t_{ijk} \vee e_{ij}$, $\neg t_{ijk} \vee e_{ik}$, $\neg t_{ijk} \vee e_{jk}$, and $\neg e_{ij} \vee \neg e_{ik} \vee \neg e_{jk} \vee t_{ijk}$. Again, the indices i, j, and k of the variable t_{ijk} may be reordered arbitrarily for notational convenience.

4.1 Encoding the Squarefree Constraint

To encode the property that a KS graph must be squarefree, we construct encodings that block all squares in the graph. Observe that three squares can be formed on four vertices. Therefore, for every choice of four vertices i, j, k, l, we use clauses $\neg e_{ij} \vee \neg e_{jk} \vee \neg e_{kl} \vee \neg e_{li}$, $\neg e_{ij} \vee \neg e_{jl} \vee \neg e_{lk} \vee \neg e_{ki}$, and $\neg e_{il} \vee \neg e_{lj} \vee \neg e_{jk} \vee \neg e_{ki}$ to encode the fact that a solution produced by the solver must be squarefree. By enumerating all possible choices of four vertices and constructing the above CNF formula, we force the graph to be squarefree. The total number of clauses needed to encode this is $3 \cdot \binom{n}{4}$.

4.2 Encoding the Minimum Degree Constraint

To ensure that vertex i is connected to at least three other vertices, we take each subset S of $\{1, \ldots, n\} \setminus \{i\}$ with cardinality $n - 3$ and construct the clause $\bigvee_{j \in S} e_{ij}$. By enumerating over all such subsets we enforce a minimum degree of 3 on vertex i. Thus, constructing similar formulae for all vertices $1 \leq i \leq n$, enforces that any vertex in the graph has a degree of at least 3. The total number of clauses needed is therefore $n \cdot \binom{n-1}{n-3} = n \cdot \binom{n-1}{2}$.

4.3 Encoding the Triangle Constraint

We encode the property that every vertex is part of a triangle as follows: for each vertex i, we require 2 other distinct vertices to form a triangle, and there are $\binom{n-1}{2}$ possible triangles containing i. At least one of those triangles must be present in the KS graph—this is encoded by the clause $\bigvee_{j,k\in S} t_{ijk}$ where S is $\{1,\dots,n\}\setminus\{i\}$ and $j<k$. Using this clause for each $1\le i\le n$ ensures that every vertex is part of a triangle and hence there are n triangle clauses.

4.4 Encoding the Noncolorability Constraint

Recall that the key property of a KS graph is that it is non-010-colorable. As stated earlier, a graph is non-010-colorable if and only if for all $\{0,1\}$-colorings of the graph, a pair of color-1 vertices is connected or a set of three color-0 vertices are mutually connected. For each $\{0,1\}$-coloring, a graph has a set V_0 of color-0 vertices and a set V_1 of color-1 vertices. Given a specific such coloring, the clause

$$\bigvee_{\substack{i,j\in V_1\\ i<j}} e_{ij} \vee \bigvee_{\substack{i,j,k\in V_0\\ i<j<k}} t_{ijk}$$

encodes that this coloring is not a 010-coloring of the graph defined by the variables e_{ij} and t_{ijk}—since either a pair of color-1 vertices is connected or three color-0 vertices are mutually connected. Note that we have to generate such a clause for all possible colorings, and conjunct them together to obtain a non-colorability constraint for graphs of order n. An assignment satisfying such a constraint corresponds to a graph that is not 010-colorable under any possible coloring. In order n, the total number of such clauses is 2^n.

An empirical observation allows cutting the number of clauses by a factor of two: $\{0,1\}$-colorings with more than $\lceil \frac{n}{2}\rceil$ color-1 vertices are unlikely to be 010-colourings, and in practice are not useful in blocking 010-colourable graphs. Put differently, by dropping the constraints with $|V_1|\ge\lceil\frac{n}{2}\rceil$ we reduce the formula size (making the formula easier to solve) and the corresponding increase in the number of satisfying assignments is small enough that these candidates can be ruled out via post-processing (Section 9). In fact, for graphs up to order 23, no additional satisfying assignments (or candidate KS graphs) were generated.

4.5 Encoding Isomorphism Blocking Clauses

Following [Codish *et al.*, 2019], we use symmetry breaking constraints enforcing a lexicographical order among the rows of the graph's adjacency matrix. These small number of additional constraints enable us to statically block many isomorphic graphs. Given an adjacency matrix A of a graph, we define $A_{i,j}$ as the ith row of A without columns i and j. Codish et al. prove that up to isomorphism every graph can be represented by an adjacency matrix A for which $A_{i,j}$ is lexicographically equal or smaller than $A_{j,i}$ for all $1\le i<j\le n$.

We express that $A_{i,j}=[x_1,x_2,\dots,x_n]$ is lexicographically equal or less than $A_{j,i}=[y_1,y_2,\dots,y_n]$ using $3n-2$ clauses and auxiliary variables $a_1,\dots,a_{n-1}$ [Knuth, 2015]. The clauses are $\neg x_k\vee y_k\vee\neg a_{k-1}$, $\neg x_k\vee a_k\vee\neg a_{k-1}$, and $y_k\vee a_k\vee\neg a_{k-1}$ for $k=1,\dots,n-1$. The literal $\neg a_0$ is omitted and the clause $\neg x_n\vee y_n\vee\neg a_{n-1}$ is also included.

5 Orderly Generation via SAT+CAS

The symmetry breaking constraints described in Section 4.5 do not block all isomorphic copies of adjacency matrices. Thus, a crucial part of the MATHCHECK pipeline is the use of a SAT+CAS combination of a SAT solver and an isomorph-free generation routine (the CAS part). The orderly isomorph-free generation approach was developed independently by [Read, 1978] and [Faradžev, 1978]. It relies on the notion of a *canonical representation* of an adjacency matrix.

Definition 8 (Canonical Graph). *An adjacency matrix M of a graph is canonical if every permutation of the graph's vertices produces a matrix lexicographically greater than or equal to M, where the lexicographical order is defined by concatenating the above-diagonal entries of the columns of the adjacency matrix starting from the left.*

An *intermediate* matrix of A is a square upper-left submatrix of A. If A is of order n then its intermediate matrix of order $n-1$ is said to be its *parent*, and A is said to be a *descendant* of its intermediate matrices. The OG method is based on the following two consequences of Definition 8:

(1) Every isomorphic class of graphs has exactly one canonical representative.

(2) If a matrix is canonical, then its parent is also canonical.

Note that the contrapositive of the second property implies that if a matrix is not canonical, then all of its descendants are not canonical. The OG process only generates canonical matrices and they are built starting from the upper-left. Therefore, any noncanonical intermediate matrix that is encountered during an OG exhaustive search can be discarded, as none of its descendants are canonical.

As described in Figure 1, in our SAT+CAS implementation, when the SAT solver finds an intermediate matrix the canonicity of this matrix is determined by a canonicity-checking routine implemented in the MATHCHECK system. If the matrix is noncanonical, then a "blocking" clause is learned which removes this matrix (and all of its descendants) from the search, thus dramatically pruning the search space. Otherwise, the matrix may be canonical and the SAT solver proceeds as normal. When a matrix is noncanonical, the canonicity-checking routine also provides a "witness" of this fact (a permutation of the vertices producing a lex-smaller adjacency matrix). We combine this process with the symmetry breaking clauses of Codish et al. that canonical matrices can be shown to satisfy [Codish *et al.*, 2019, Def. 8].

The OG technique provides a speedup that seems to increase exponentially in the order n of the KS graph—see Table 2, which provides experimental running times comparing the SAT+CAS approach against SAT-only and CAS-only approaches. The CAS compared against was the *nauty* graph generator [McKay and Piperno, 2014] with the same configuration from [Uijlen and Westerbaan, 2016]. More details on our experimental setup can be found in Section 8.

Further, as described in Figure 1, we simplify the SAT instance using the SAT solver CaDiCaL [Biere *et al.*, 2020] before solving the instance using MapleSAT [Liang *et al.*, 2016]. We implemented the OG technique in both solvers (using IPASIR-UP in CaDiCaL [Fazekas *et al.*, 2023]), and thus both the simplification and solving incorporate the knowledge derived from OG.

n	SAT+CAS	SMS	CAS-only	SAT-only
17	0.3 m	0.2 m	25.2 m	9.0 m
18	1.8 m	1.2 m	455.4 m	266.4 m
19	9.0 m	8.4 m	9506.4 m	11,705.8 m
20	140.5 m	100.8 m	timeout	timeout
21	1945.0 m	1574.4 m	timeout	timeout

Table 2: **SAT+CAS vs. SMS, CAS-only (nauty), and SAT-only:** The total CPU time (in minutes) compared on orders $17 \leq n \leq 21$. All tools are sequential. For higher orders, CAS-only and SAT-only time out at 12,000 minutes.

6 Embeddability Checking

Every KS candidate must be embeddable in order to qualify as a KS graph. Hence, we perform an embeddability check on every KS candidate generated by the SAT+CAS solver of MathCheck. Operationally, a graph G is said to be embeddable when every pair of adjacent G-vertices can be mapped to two orthogonal vectors on the unit sphere in $\mathbf{R}^3$.

Our embeddability checking algorithm consists of two parts. The first part is an integration of the vector assignment algorithm that finds all possible vector assignments describing the orthogonal relations between the vectors v_i in a KS candidate defined by a set of edges [Uijlen and Westerbaan, 2016]. The second part of the algorithm applies an SMT solver to determine the satisfiability of a system of nonlinear equations generated from a particular vector assignment as described below.

More precisely, an assignment generated by Uijlen and Westerbaan's algorithm is converted into a set of cross and dot product equations, and these equations are passed to the theorem prover Z3 [de Moura and Bjørner, 2008] to solve the equations over the real numbers.

To check whether a graph is embeddable, we use Z3 to determine whether these nonlinear arithmetic constraints are satisfiable over the real numbers. If a solution is found, it is an assignment of vertices to vectors satisfying all orthogonality constraints and the graph is therefore embeddable. We further optimized the pipeline by precomputing minimal unembeddable graphs, as defined below.

Definition 9 (Minimal Unembeddable Graph). *A graph G is said to be a **minimal unembeddable graph** if any proper subgraph of G is embeddable, but G itself is unembeddable.*

A graph is unembeddable if and only if it contains a minimal unembeddable subgraph. To optimize embeddability checking, we precomputed all minimal unembeddable graphs of orders up to and including 12. It was only necessary to consider squarefree graphs with a minimum vertex degree of 2, as the square graph C_4 is minimally unembeddable itself, and a graph containing a vertex of degree 0 or 1 is not minimally unembeddable.

We implement the unembeddable subgraph blocking technique as part of MathCheck. If a graph contains one of the 17 minimal unembeddable graphs up to order 12 as a subgraph, the graph is blocked dynamically and is not considered as a satisfying assignment by MathCheck. This technique leads to a significant reduction of the number of satisfying assignment (candidates) generated by the SAT+CAS solver.

	Number of Cubes		Total CPU time	
n	SAT+CAS	SMS	SAT+CAS	SMS
22	26,646	18,659	932 h	628 h
23	173,097	313,665	12,116 h	11,922 h

Table 3: **Parallel CnC SAT+CAS vs. parallel SMS:** The number of cubes and total CPU time for the parallel versions of SAT+CAS and the SAT Modulo Symmetries tool on orders $22 \leq n \leq 23$.

7 Parallelization

In prior applications of the CnC technique [Heule *et al.*, 2017; Heule *et al.*, 2016], the cubing solver generates a collection of cubes before the conquering solver is invoked. Subsequently, each of these subproblems is solved using the conquering solver in parallel. However, this approach presents two primary challenges. Firstly, the generated cubes might exhibit imbalanced solving times, especially since the cubing solver does not have the ability to call the CAS to incorporate isomorph-free generation. Secondly, the proof size for each subproblem also varies, making it difficult to allocate an appropriate amount of memory to individual cores. In MathCheck, we implement a slight modification of traditional CnC practices to resolve the above challenges. In addition, we use a new CnC solver AlphaMapleSAT [Jha *et al.*, 2024] that provides significant speedup for the cubing process.

In our proposed method, the cubing solver of AlphaMapleSAT operates on a much smaller CNF instance obtained by omitting all non-colorability constraints (Section 4.4). This approach is substantiated by empirical evidence suggesting that the same set of cubes can be obtained even without the non-colorability constraints. AlphaMapleSAT iteratively generates cubes until the total number of cubes surpasses a predefined cutoff, which is based on available computational resources.

Following cube generation, each subproblem is processed by the simplification solver (CaDiCaL with OG), and then it is passed to the conquering solver (MapleSAT with OG) and solved in parallel. To efficiently manage the termination of each subproblem, a termination strategy is implemented: if the proof size for a subproblem exceeds 7 GiB, the problem is further divided into more cubes and solved accordingly. This iterative process continues until every subproblem can be resolved with proof sizes under 7 GiB. We overcome the challenges posed by varying proof sizes by implementing this slight modification and it allows us to verify all generated proof certificates with at most 4 GiB of memory allocation.

8 Experimental Results

Results in Table 2 were conducted on a cluster of Intel E5-2683 v4 Broadwell @ 2.1GHz CPUs, each with access to 4 GiB of RAM and running 64-bit CentOS Linux 7. Results in Table 3 were conducted on a cluster of Dual Intel Xeon Gold 6226 CPUs @ 2.7 GHz (24 cores/node). We used the `g++` compiler version 9.3.0 with option `-O3` to compile the SAT solvers used. 24 CPUs were used to solve and verify order 22 (in about 4 days wall clock time), and 240 CPUs were used for order 23 (in about 7 days wall clock time).

To compare MATHCHECK against other approaches, we computed the total CPU time used by all processes in the solving process, including simplification, cubing, SAT solving, CAS queries, and embeddability checking.

8.1 Key Findings

Tables 2 and 3 contain our experimental results. As can be seen from Table 2, the sequential version of MATHCHECK outperforms the sequential SAT-only (MapleSAT) and CAS-only (nauty) tools. Further, the sequential MATHCHECK is comparable to sequential SMS on orders 17 to 21, even with proof production switched on. (Note that we are using the results reported by the SMS authors [Kirchweger *et al.*, 2023] which did not include generating proofs.)

The sequential MATHCHECK also outperforms the parallel tool of [Uijlen and Westerbaan, 2016]. For example, for order 21, sequential MATHCHECK takes about 32.4 hours while their tool took 3 months on 300 CPUs in 2014 (the exact kind of CPUs used are not specified). Further, their tool did not solve orders 22 or greater due to time constraints.

For higher orders of KS (i.e., 22 and 23), we do not expect the sequential versions of MATHCHECK and SMS to scale in a reasonable amount of time, and hence we compare only their parallel CnC versions (Table 3). The authors of SMS also do not report on the performance of their sequential version for orders 22 and 23. In order 22, SMS is about $1.5\times$ faster than SAT+CAS, while in order 23, the two approaches have comparable runtime.

Our approach was hindered due to our generating and verifying all proof certificates—requiring us to ensure each proof certificate did not grow too large in order to permit effective verification. Such an approach requires splitting each cube whose proof certificate grows too large, and this additional splitting slows down the solver.

Our method benefits from dynamically blocking minimal unembeddable subgraphs up to order 12, which enables MATHCHECK to generate significantly fewer KS candidates compared to Kirchweger *et al.* Specifically, we generate a single candidate in order 22 and 41 candidates in order 23, while Kirchweger *et al.* generates 88,282 and 3,747,950 candidates respectively.

Another notable question we address is the existence of a complex KS vector system. Specifically, is there a KS graph that can be embedded over the complex sphere? We employ a similar embeddability verification pipeline as mentioned earlier, with the distinction that vectors now may possess complex number coordinates. We discover that each KS graph up to order 23 either contains a complex unembeddable subgraph or is determined to be unembeddable over the complex numbers via Z3. Therefore, we conclude that the minimum size for both the real and complex KS systems are at least 24.

9 Verification of Results

In order to verify the computations produced by MATHCHECK, we use DRAT proof logging which is standard in all modern SAT solvers. This makes it possible for a third-party proof verifier to provide an independent certification of the correctness of the solver's conclusion (assuming the input Boolean formula correctly encodes the minimum KS problem).

A DRAT proof consists of a trace of the clauses learned by the solver during its execution. A proof verifier checks that each clause can be derived from the previous clauses using simple rules known to be logically consistent. The CAS-derived noncanonical blocking clauses cannot be verified using the normal rules, so we specially tag them to be verified separately. Specifically, they are justified via a CAS-derived permutation that, when applied to the blocked adjacency matrix, produces a lex-smaller adjacency matrix—and therefore provides a witness that the blocked matrix is noncanonical and is safe to block. Similarly, the unembeddable subgraph blocking clauses are justified by a CAS-derived permutation that when applied to the blocked subgraph transforms it into one of the minimal unembeddable subgraphs (see Section 6).

The CAS-derived clauses in the DRAT proof are prefixed by the character '`t`' to signify they should be trusted and we modified DRAT-trim [Wetzler *et al.*, 2014] to trust such clauses following the approach first used in [Bright *et al.*, 2020a]. The trusted CAS-derived clauses are separately verified by a permutation-applying Python script that applies the witnesses produced by the CAS to verify the blocked matrices are noncanonical or unembeddable.

Similarly, when a KS candidate is found, the solver learns a trusted clause blocking the candidate (so that the search continues until all candidates have been found). The DRAT proof ends with the empty clause, which by definition is not satisfiable. If the verifier is indeed able to verify the empty clause, then we can have confidence that the SAT solver's search missed no candidates without needing to trust the solver.

All results have been certified. The uncompressed proofs for order 22 are about 6.1 TiB in total and 40.3 TiB in order 23. The certification for orders 22 and 23 required using CnC (see Section 7) to ensure that each DRAT proof could be verified with at most 4 GiB of memory.

All KS candidates produced by our method have been extensively checked. For example, each KS candidate is passed into a verification script implemented using the NetworkX [Hagberg *et al.*, 2008] graph package to verify that they satisfy all encoded constraints (see Section 4).

Further, we test the embeddability pipeline by performing verification on all squarefree graphs of order up to 12 with minimal vertex degree 2. Specifically, if a graph is embeddable and corresponds to a set of vectors, we check that no pair of vectors in the set are collinear, and a pair of vectors are orthogonal if their corresponding vertices are connected.

10 Conclusion

We present a new proof-producing SAT+CAS method, implemented in MATHCHECK, using orderly generation to give a computer-assisted verifiable proof showing that the minimum KS vector system in 3D contains at least 24 vectors. Further, our sequential version is orders of magnitude faster than SAT-only, CAS-only (nauty), and prior CAS-only methods, and our parallel version is comparable to a SAT modulo symmetries approach.

References

[Ábrahám, 2015] Erika Ábrahám. Building bridges between symbolic computation and satisfiability checking. In *Proceedings of the 2015 ACM on ISSAC*, pages 1–6, 2015.

[Aloul *et al.*, 2003] Fadi A. Aloul, Karem A. Sakallah, and Igor L. Markov. Efficient symmetry breaking for Boolean satisfiability. In *Proceedings of IJCAI*, IJCAI'03, page 271–276, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[Arends *et al.*, 2011] Felix Arends, Joël Ouaknine, and Charles W Wampler. On searching for small Kochen-Specker vector systems. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 23–34. Springer, 2011.

[Barrett *et al.*, 2016] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

[Biere *et al.*, 2020] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53, Helsinki, 2020. University of Helsinki.

[Biere *et al.*, 2021] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, Amsterdam, February 2021.

[Bright *et al.*, 2016] Curtis Bright, Vijay Ganesh, Albert Heinle, Ilias Kotsireas, Saeed Nejati, and Krzysztof Czarnecki. MATHCHECK2: A SAT+CAS verifier for combinatorial conjectures. In *International Workshop on Computer Algebra in Scientific Computing*, pages 117–133. Springer, 2016.

[Bright *et al.*, 2020a] Curtis Bright, Kevin K. H. Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. Nonexistence certificates for ovals in a projective plane of order ten. In *Lecture Notes in Computer Science*, volume 12126, pages 97–111. Springer International Publishing, Cham, 2020.

[Bright *et al.*, 2020b] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. Applying computer algebra systems with SAT solvers to the Williamson conjecture. *JSC*, 100:187–209, 2020.

[Bright *et al.*, 2021] Curtis Bright, Kevin K. H. Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. A SAT-based resolution of Lam's problem. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 3669–3676, California USA, 2021. AAAI Press.

[Bright *et al.*, 2022] Curtis Bright, Ilias Kotsireas, and Vijay Ganesh. When satisfiability solving meets symbolic computation. *Communications of the ACM*, 65(7):64–72, July 2022.

[Cañas *et al.*, 2014] Gustavo Cañas, Mauricio Arias, Sebastián Etcheverry, Esteban S Gómez, Adán Cabello, Guilherme B Xavier, and Gustavo Lima. Applying the simplest Kochen-Specker set for quantum information processing. *Physical review letters*, 113(9):090404, 2014.

[Codish *et al.*, 2019] Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1):1–24, August 2019.

[Conway and Kochen, 2002] John Conway and Simon Kochen. *The Geometry of the Quantum Paradoxes*, page 257–269. Springer Berlin Heidelberg, 2002.

[Conway and Kochen, 2006] John Conway and Simon Kochen. The free will theorem. *Foundations of Physics*, 36(10):1441–1473, July 2006.

[Crawford *et al.*, 1996] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, page 148–159, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340. Springer, 2008.

[Faradžev, 1978] I A Faradžev. Constructive enumeration of combinatorial objects. In *Problèmes combinatoires et théorie des graphes*, pages 131–135, 1978.

[Fazekas *et al.*, 2023] Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: user propagators for CDCL. In *SAT 2023*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[Ganesh and Vardi, 2020] Vijay Ganesh and Moshe Y. Vardi. *On the Unreasonable Effectiveness of SAT Solvers*, page 547–566. Cambridge University Press, December 2020.

[Gerlach and Stern, 1922] Walther Gerlach and Otto Stern. Der experimentelle nachweis der richtungsquantelung im magnetfeld. *Zeitschrift für Physik*, 9:349–352, 1922.

[Hagberg *et al.*, 2008] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[Held, 2000] Carsten Held. The Kochen-Specker Theorem. https://plato.stanford.edu/entries/kochen-specker/, 2000. Accessed: 2024-05-28.

[Heule *et al.*, 2016] Marijn J H Heule, Oliver Kullmann, and Victor W Marek. Solving and verifying the boolean Pythagorean triples problem via cube-and-conquer. In *SAT Conference*, pages 228–245, Cham, 2016. Springer.

[Heule *et al.*, 2017] Marijn J H Heule, Oliver Kullmann, and Victor W Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In *IJCAI*, volume 17, pages 228–245, 2017.

[Heule, 2019] Marijn J. H. Heule. Optimal symmetry breaking for graph problems. *Mathematics in Computer Science*, 13(4):533–548, May 2019.

[Huang *et al.*, 2003] Yun-Feng Huang, Chuan-Feng Li, Yong-Sheng Zhang, Jian-Wei Pan, and Guang-Can Guo. Experimental test of the Kochen-Specker theorem with single photons. *Physical Review Letters*, 90(25), June 2003.

[Jha *et al.*, 2024] Piyush Jha, Zhengyu Li, Zhengyang Lu, Curtis Bright, and Vijay Ganesh. Alphamaplesat: An MCTS-based cube-and-conquer SAT solver for hard combinatorial problems. *arXiv preprint arXiv:2401.13770*, 2024.

[Jost, 1976] Res Jost. Measures on the finite dimensional subspaces of a Hilbert space: remarks to a theorem by A. M. Gleason. *Studies in Mathematical Physics: Essays in Honour of Valentine Bergmann*, pages 209–228, 1976.

[Junttila *et al.*, 2020] Tommi Junttila, Matti Karppa, Petteri Kaski, and Jukka Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *JSC*, 99:21–49, July 2020.

[Kaufmann and Biere, 2023] Daniela Kaufmann and Armin Biere. Improving AMulet2 for verifying multiplier circuits using SAT solving and computer algebra. *International Journal on Software Tools for Technology Transfer*, January 2023.

[Kirchweger and Szeider, 2021] Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In *CP 2021*, volume 210, pages 34:1–34:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl.

[Kirchweger *et al.*, 2023] Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with sat modulo symmetries. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023. International Joint Conferences on Artificial Intelligence Organization, August 2023.

[Knuth, 2015] Donald E Knuth. *The art of computer programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, Massachusetts, 2015.

[Kochen and Specker, 1967] Simon Kochen and E. P. Specker. The Problem of Hidden Variables in Quantum Mechanics. *JMM*, 17:59–87, 1967.

[Li *et al.*, 2022] Zhengyu Li, Curtis Bright, and Vijay Ganesh. An SC-Square approach to the minimum Kochen–Specker problem. In Ali Kemal Uncu and Haniel Barbosa, editors, *Proceedings of the 7th SC-Square Workshop*, volume 3458 of *CEUR Workshop Proceedings*, pages 55–66. CEUR-WS.org, 2022.

[Liang *et al.*, 2016] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *SAT 2016 - 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings*, pages 123–140, 2016.

[Mahzoon *et al.*, 2022] Alireza Mahzoon, Daniel Große, Christoph Scholl, Alexander Konrad, and Rolf Drechsler. Formal verification of modular multipliers using symbolic computer algebra and boolean satisfiability. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, New York, July 2022. ACM.

[McKay and Piperno, 2014] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *JSC*, 60:94–112, 2014.

[Metin *et al.*, 2018] Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. CDCLSym: Introducing effective symmetry breaking in SAT solving. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 99–114, New York, 2018. Springer International Publishing.

[Neiman *et al.*, 2022] David Neiman, John Mackey, and Marijn Heule. Tighter bounds on directed Ramsey number $R(7)$. *Graphs and Combinatorics*, 38(5), September 2022.

[Pavičić *et al.*, 2005] Mladen Pavičić, Jean-Pierre Merlet, Brendan McKay, and Norman D. Megill. Kochen–Specker vectors. *Journal of Physics A: Mathematical and General*, 38(7):1577–1592, February 2005.

[Pavičić and Megill, 2022] Mladen Pavičić and Norman D. Megill. Automated generation of arbitrarily many Kochen-Specker and other contextual sets in odd-dimensional Hilbert spaces. *Physical Review A*, 106(6), December 2022.

[Peres, 1991] Asher Peres. Two simple proofs of the Kochen–Specker theorem. *Journal of Physics A: Mathematical and General*, 24(4):L175, 1991.

[Read, 1978] Ronald C Read. Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. *Annals of Discrete Mathematics*, 2:107–120, 1978.

[Savela *et al.*, 2020] Jarkko Savela, Emilia Oikarinen, and Matti Järvisalo. Finding periodic apartments via Boolean satisfiability and orderly generation. In *EPiC Series in Computing*, UK, 2020. EasyChair.

[Sellmann and Hentenryck, 2005] Meinolf Sellmann and Pascal Van Hentenryck. Structural symmetry breaking. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05*, pages 298–303, California, 2005.

[Uijlen and Westerbaan, 2016] Sander Uijlen and Bas Westerbaan. A Kochen-Specker system has at least 22 vectors. *New Generation Computing*, 34(1):3–23, 2016.

[Wetzler *et al.*, 2014] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Lecture Notes in Computer Science*, volume 8561, pages 422–429. Springer International Publishing, Cham, 2014.

[Zulkoski *et al.*, 2015] Edward Zulkoski, Vijay Ganesh, and Krzysztof Czarnecki. MathCheck: A math assistant via a combination of computer algebra systems and SAT solvers. In *CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 607–622, Cham, 2015. Springer.