An End-To-End Verification of Keller's Conjecture

James Gallicchio
Carnegie Mellon University, Pittsburgh, PA, USA

Cayden Codel
Carnegie Mellon University, Pittsburgh, PA, USA

Jeremy Avigad
Carnegie Mellon University, Pittsburgh, PA, USA

Marijn J. H. Heule
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

In 1930, Keller conjectured that every gap-free tiling of $\mathbb{R}^n$ by n-dimensional unit cubes must contain cubes that fully share an $(n-1)$-dimensional face. Keller's conjecture holds for $n \leq 7$ and fails for $n \geq 8$. The final case, $n = 7$, was settled in 2020 using a mix of traditional and automated reasoning. The result was obtained by reducing the conjecture to a set of clique-existence problems, encoding those problems into propositional logic, breaking symmetries, and solving them with a SAT solver.

In this paper, we present an end-to-end verification in Lean 4 of Keller's conjecture for all dimensions. First, we simplify a prior reduction of Keller's conjecture to the clique-existence problems. We then verify an improved SAT encoding of those problems, as well as some symmetry reasoning on the encoding. Throughout our work, we sought to maximize the synergy between interactive and automated techniques while minimizing human proof burden. In particular, the symmetry reasoning was split between Lean and a mechanically-checkable proof system, since neither was suitable on their own for verifying all of the symmetry reasoning. We discuss how and why we chose to split the reasoning across these systems based on their relative strengths and weaknesses.

2012 ACM Subject Classification Theory of computation → Logic and verification; Mathematics of computing → Graph theory

Keywords and phrases Keller's conjecture, the Lean theorem prover, SAT encodings, SAT solving, Trestle, formal verification

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.26

Supplementary Material *Software*: `https://github.com/FormalSAT/trestle`
archived at `swh:1:dir:6b4a536d8e958f97f8c03173ac615ab97fa57f84`

Funding This research was partially supported by the DARPA expMath program through DARPA CMO contract number HR0011262E028, by the NSF AIMing program through NSF Grant DMS-2434614, and by the Hoskinson Center for Formal Mathematics.

Declaration about GenAI/LLM use We did not use LLMs or other sources of AI in the creation of our formalization, and aside from the occasional question to an LLM chatbot about Tikz styling, we did not use LLMs or other sources of AI in the writing of this paper.

1 Introduction

In 1930, Ott-Heinrich Keller conjectured that every tiling of the n-dimensional space by unit cubes must contain adjacent cubes that fully share an $(n-1)$-dimensional face. Intuitively, a *tiling* is a set of cubes that are non-overlapping, gap-free, and together cover $\mathbb{R}^n$. For example, all tilings of the plane by unit squares must contain two squares that share an edge (see Figure 1). Keller's conjecture has an interesting history involving broader connections between abstract algebra and geometry; see Debroni et al. [15] for a wonderful overview.

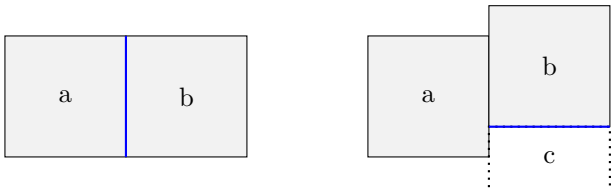


Figure 1 A visual proof that Keller's conjecture holds for $n = 2$. Either squares (a) and (b) share an edge, or the tiling must include square (c), which shares an edge with (b).

By 2002, the conjecture was known to be true for $n \leq 6$ [15, 30] and false for $n \geq 8$ [12, 23, 33], leaving open the $n = 7$ case. Incremental results [21, 22, 36] eventually led to a proof in 2020 due to Brakensiek, Heule, Mackey, and Narváez (BHMN) [6]. By employing a mix of pen-and-paper proofs and automated reasoning tools, BHMN showed that the conjecture holds for $n = 7$, thus fully resolving the conjecture.

The mix of traditional and automated reasoning used by BHMN is becoming an increasingly common mathematical tool. For example, similar proof strategies were used to resolve the Pythagorean triples problem [18], Lam's problem [7], and the empty hexagon problem [20]. In general, the strategy is to reduce the problem to the satisfiability of a particular set of propositional formulas, which are then solved by a boolean satisfiability (SAT) solver. Often, these reductions involve clever encodings, symmetry arguments, and case splits to make the formulas tractable to solve. However, the use of these techniques can also potentially introduce errors into the reduction, and as a result several reductions have received formalizations in their own right, including for the Pythagorean triples [14] and empty hexagon problems [32], with the latter being one of the first true end-to-end formalizations with a significant computational portion resolved by a SAT solver.

The formalization of Keller's conjecture has followed a similar journey, albeit a slightly longer one. BHMN reduced the conjecture to the existence of a clique in a particular family of graphs, which was then encoded into SAT. Then they added symmetry-breaking constraints to the formulas to make them more tractable to solve. Some of the symmetry reasoning was handled by a mechanical proof system called *substitution redundancy* (SR) [11], but everything else was proven using pen-and-paper proofs. In 2023, Joshua Clune [9] formalized the clique reduction in the Lean 3 theorem prover, but he left open the verification of the SAT encoding and the symmetry reasoning. In this paper, we finally close this gap.

Contributions. We present the first end-to-end verification of Keller's conjecture for all dimensions, which we completed using the Lean 4 theorem prover [28]. In addition to updating and significantly shortening Clune's formalization, our contributions are as follows:

- We improved and verified BHMN's SAT encoding of the clique-existence problem.
- We verified BHMN's symmetry reasoning, partially in Lean and partially in SR.
- We improved BHMN's case split to reduce SAT-solving costs.
- We used a verified proof checker [11] to formally check the results from the SAT solver.

Overall, our formalization shows that it is possible to effectively mix manual and automated verification in a single project without compromising correctness, and we will discuss how the different costs of interactive versus automated methods affected our formalization.

2 Reducing Keller to Cliques

In this section, we define Keller's conjecture both on paper and in Lean, and we discuss how the conjecture reduces to the existence of cliques in the so-called *Keller graphs*. While several variants of this reduction exist in the literature, we focus on the one due to BHMN [6], which was ultimately formalized by Clune [9].

Our formalization follows Clune's, although we make several simplifications. In the end, our version comprises about 3,000 lines of Lean 4 code (including comments and whitespace) and represents about 1 month of work. For comparison, Clune's formalization comprises about 15,000 lines of Lean 3 code. While part of this disparity can be attributed to the continued development of `mathlib4` [26][1] and to improvements in Lean's proof automation, our choice to use *Keller colorings* instead of Keller graphs (see Section 3) eliminated many lemmas and subgoals on integer arithmetic that Clune's formalization had to grapple with. Overall, this portion of our work confirms the theme in formal verification that the right abstractions and proof-automation tools can substantially reduce the costs of formalization.

Keller's conjecture is usually expressed in terms of Euclidean geometry: that every tiling of $\mathbb{R}^n$ with n-dimensional unit cubes must contain a pair of adjacent cubes that completely share an $(n-1)$-dimensional face. A *tiling* T is a set of cubes that are axis-aligned, non-overlapping, gap-free, and together cover $\mathbb{R}^n$, meaning that every point $p \in \mathbb{R}^n$ is contained in a unique cube $c \in T$. In the literature, it is conventional to identify $T \subset \mathbb{R}^n$ with the set of minimum *corners* of each cube in the tiling. To prevent double counting along (partially) shared faces, the cube extending from c contains only the faces that intersect with c. In other words, the cube defined by c is the half-open interval $[0,1)^n$ translated by c, which we write as $c + [0,1)^n$. Two cubes are *facesharing* if their corners differ by a unit vector along some axis while being equal along all other axes. A tiling is *faceshare-free* if no pair of cubes are facesharing, and such a tiling would show that the conjecture is false for dimension n.

These definitions are relatively straightforward to state in Lean; see Figure 2. Note that we define points in $\mathbb{R}^n$ as functions, rather than as vectors or lists of length n, since functions compose better in Lean. This means we access the dth coordinate of a point p in Lean using function application `p d`, rather than writing p_d or $p[d]$.

From here, the goal is to reduce Keller's conjecture to the existence of a large clique in a family of graphs. This reduction, pictured in Figure 3, follows Appendix A from BHMN's paper and was formalized by Clune. Since we repeat prior work, we give only a proof sketch.

A central concept of the reduction is the *core* of a tiling T. First, observe that every cube with corner $c \in \mathbb{R}^n$ contains a unique integer-valued point $p \in \mathbb{Z}^n$, which we call the *index* of the cube. Since the reverse is also true, i.e., every index is contained in a unique cube in T, there exists a bijection between T and $\mathbb{Z}^n$. The *core* of T is the set of cubes whose index is in the set $I^n \coloneqq \{0,1\}^n$. Visually, these are the 2^n cubes in T that intersect the box $[0,1]^n$, with a unique cube covering each corner of the box.

The first half of the reduction shows that Keller's conjecture holds for arbitrary cube tilings if and only if it holds for *periodic* cube tilings. A tiling T is *periodic* if the cubes in T are closed under translation by $2z$ for any $z \in \mathbb{Z}^n$. In other words, shifting the entire tiling by an integer multiple of 2 units along any axis maps the tiling back to itself. As it turns out, the core of T can be used to construct a new periodic tiling T' by translating the core by $2z$ for every $z \in \mathbb{Z}^n$. Visually, this means taking the cubes in T that intersect I^n and tessellating them by 2 units along every axis. Notably, if T is faceshare-free, then so is T'.

[1] `https://github.com/leanprover-community/mathlib4`

```
abbrev Point (n : ℕ) := Fin n → ℝ
def UnitCube (n : ℕ) : Set (Point n) := { p | ∀ d, 0 ≤ p d ∧ p d < 1 }
def Cube (corner : Point n) : Set (Point n) := (corner + ·) '' (UnitCube n)

structure Tiling (n : ℕ) where
  corners : Set (Point n)
  covers : ∀ p : Point n, ∃! c ∈ corners, p ∈ Cube c

def Faceshare (c₁ c₂ : Point n) : Prop :=
  ∃ d, |c₁ d - c₂ d| = 1 ∧ ∀ d' ≠ d, c₁ d' = c₂ d'

def Tiling.FaceshareFree (T : Tiling n) : Prop :=
  T.corners.Pairwise (fun p₁ p₂ => ¬ Faceshare p₁ p₂)

def conjectureIn (n : ℕ) : Prop := ¬ ∃ (T : Tiling n), T.FaceshareFree
```

Figure 2 The Lean definitions for Keller's conjecture. `Fin n` is the type of natural numbers less than `n`. `Cube` is defined as the translation of the `UnitCube` $[0,1)^n$ by a `corner` point. The syntax `f '' S` means the image of the set `S` under `f`. The quantifier "$\exists!\, x,\, P(x)$" means "there exists a unique x such that $P(x)$ holds.".

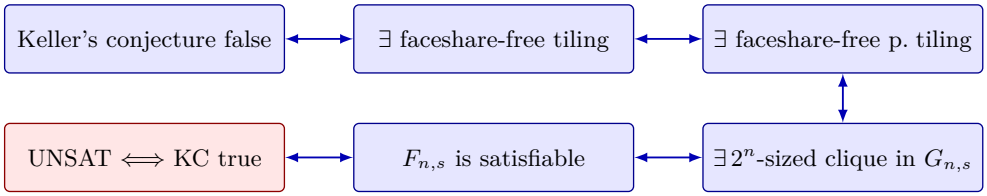


Figure 3 The proof sketch, due to BHMN and formalized by Clune, reducing Keller's conjecture in dimension n to a SAT problem. The boxes are related by if-and-only-if relations. The red box indicates that if the formula is UNSAT, then Keller's conjecture holds for that dimension.

The second half of the reduction shows that there exist faceshare-free periodic tilings if and only if there exist cliques of size 2^n in certain Keller graphs. A *Keller graph* $G_{n,s}$ is a graph with vertices the length-n $2s$-ary vectors, written as $\langle 2s\rangle^n$, where $\langle 2s\rangle \coloneqq \{0, 1, \ldots, 2s-1\}$. Two vertices are adjacent in the Keller graph if they: (1) differ by s in some coordinate, and (2) differ in some other coordinate. If K is a 2^n-sized clique in $G_{n,s}$, then a faceshare-free tiling T can be constructed from K using the points $\{\frac{v}{s} \mid v \in K\}$ as the periodic core of T. Intuitively, condition (1) ensures that all cube corners are at least 1 unit distance apart, and condition (2) ensures that possibly-facesharing corners are not actually facesharing.

BHMN showed that Keller's conjecture is false for dimension n if and only if a 2^n-sized clique exists in $G_{n,s}$ for $s = 2^{n-1}$. However, since this graph is somewhat large, they used discretization results for the $n = 7$ case due to Kisielewicz and Łysakowska [21, 22, 36] to show that it is sufficient to check for 2^7-sized cliques in the graphs $G_{7,s}$ for $s \in \{3, 4, 6\}$.[2]

From there, BHMN wrote a propositional formula expressing the existence of a 2^7-sized clique in $G_{7,s}$ for each $s \in \{3, 4, 6\}$, and then they used a SAT solver to prove the non-existence of the cliques, thus showing that Keller's conjecture is true for $n = 7$.

[2] For technical reasons, it is sufficient to check only $s = 6$, but BHMN checked all three cases regardless.

Clune [9] formalized the reduction of Keller's conjecture to the existence of a clique in $G_{n,s}$ for $s = 2^{n-1}$, but he did not formalize Kisielewicz's and Łysakowska's discretization results, nor did he formalize the SAT encoding, symmetry reasoning, or SAT-solving process used to resolve the $n = 7$ case, leaving a formalization gap. In fact, at the end of his paper, Clune remarked that closing this gap would mean either formalizing the discretization results (which would involve a lot of complicated mathematics) or improving the SAT encoding so that the $G_{7,64}$ case could be tractably solved by a SAT solver.[3] We chose to do the latter.

3 Coloring Problem

One of our contributions is reframing the clique-existence problem as a coloring problem. In the clique version, we must constantly show that two vertices differ by s in some coordinate, which involves tedious arithmetic in Lean. In the coloring version, we only have to show that a coloring function is (un)equal on various inputs, which is much easier to formalize.

The coloring problem arises from the particular structure of Keller graphs. In 2011, Debroni et al. [15] observed that the Keller graph $G_{n,s}$ can be partitioned into 2^n independent sets. See Figure 4 for an illustration. If we use the tiling construction $\{\frac{v}{s} \mid v \in K\}$ discussed at the end of Section 2, then the vertices of $G_{n,s}$ correspond to the $(2s)^n$ points in $[0, 2)^n$ whose coordinate values are integer multiples of $\frac{1}{s}$, and the s^n points contained in the unit cube c_i extending from any *index*[4] in $I^n \coloneqq \{0, 1\}^n$ form each independent set, since no two points will be exactly 1 unit apart in any coordinate. For example, if $n = 2$ and $s = 2$, then the four points in the cube c_i extending from $i = (1, 0)$, i.e., $(1, 0)$, $(\frac{3}{2}, 0)$, $(1, \frac{1}{2})$, and $(\frac{3}{2}, \frac{1}{2})$, form an independent set. Notationally, we will sometimes write indices i as least-significant-bit-first length-n binary strings, and we will refer to their (0-indexed) dth component as i_d. For example, if $n = 7$, then the index 5 can be written as the string $i = 1010000$, and e.g. $i_2 = 1$. Two indices i and j are *adjacent* if they share an edge in the hypercube defined by I^n, meaning their binary string representations differ by a single bit. Note that index adjacency is distinct from, but related to, vertex adjacency in Keller graphs.

Given these independent sets, forming a clique in a Keller graph means picking a point in the unit cube c_i for every index $i \in I^n$. Equivalently, for each index i, we pick an integer multiple of $\frac{1}{s}$ in $[0, 1)$ for each coordinate, or rather, we assign one of s *colors* to each coordinate. Figure 5 illustrates a partial coloring of the indices.

Putting these ideas together, we can restate the clique problem as a coloring problem.

▶ **Definition 1** (Keller coloring). *A* Keller coloring *in dimension n with s colors is a mapping $C : I^n \to \langle s \rangle^n$ from indices to length-n s-ary color vectors such that the following hold:*

1. **(Gap)** *For all indices $i \neq j \in I^n$, there exists some dimension d for which the indices differ and the color vectors are the same: $\exists\, d,\ i_d \neq j_d \ \wedge\ C(i)_d = C(j)_d$.*
2. **(No-Faceshare)** *All pairs of adjacent indices i and j have pairwise different color vectors: $C(i) \neq C(j)$, or equivalently, $\exists d,\ C(i)_d \neq C(j)_d$.*

To further see how colorings are an equivalent formulation, consider the following. Keller colorings C are in bijection with cliques $K \subset V(G_{n,s})$ via $K \coloneqq \{v(i) \mid i \in I^n\}$, where $v(i)_d \coloneqq s * i_d + C(i)_d$, and the coloring conditions correspond to the vertex-adjacency

[3] Alternatively, one could find a SAT-less proof that there are no 2^7-sized cliques in $G_{7,64}$, but to the best of our knowledge such a proof is not forthcoming.

[4] The careful reader will notice that each index $i \in I^n$ is its own index, using the prior definition of an index as the unique integer-valued point in any unit cube. We re-use the term "index" on purpose.

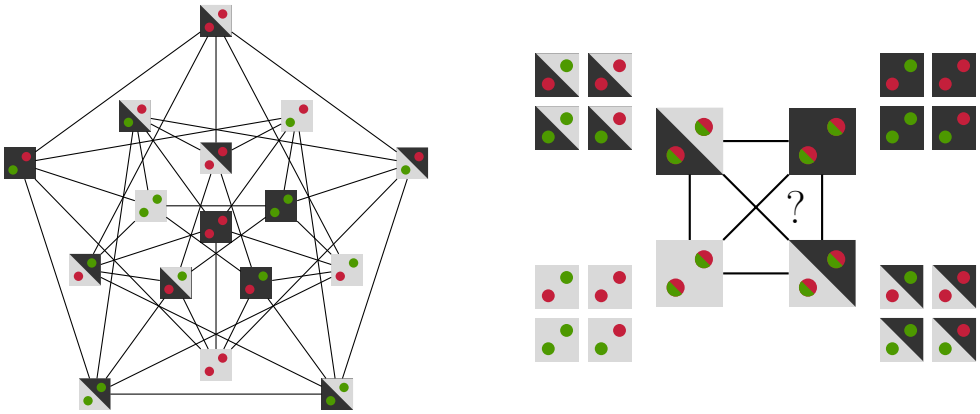


Figure 4 The Keller graph $G_{2,2}$ (left) and a partitioning of its vertices into 4 (more generally, 2^n) independent sets (right). Keller's conjecture is false if the graph has a clique of size $2^n = 4$, and such a clique must contain exactly one vertex from each independent set. The vertices are depicted here as "dice" with $n = 2$ pips. Each pip corresponds to a different entry in that vertex's length-n $2s$-ary vector, with the 2 background colors and $s = 2$ dot colors corresponding to the $2s = 4$ possible values for each entry. Two vertices are adjacent in the graph if one pair of corresponding pips differs only in their background color and the other (or, for $n > 2$, any other) pair of pips differs in any way.

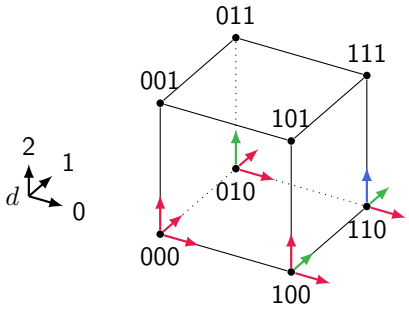


Figure 5 A partial Keller coloring on the indices of I^3, depicted as the corners of a (hyper)cube. Each index receives one of s colors for each coordinate. Here, only half of I^3 has been colored.

conditions in the Keller graph: The Gap condition ensures that every pair of vertices has a coordinate d where their vectors differ by s, and the No-Faceshare condition ensures that the vectors of adjacent indices differ in at least one other coordinate. Note that non-adjacent indices map to vertices with two differences no matter the coloring.

We visualize (partial) colorings for certain indices as in Figure 6. Because the binary bits of the index have semantic meaning, we depict the bits with light or dark backgrounds, while the numbers in each square represent the colors assigned to each coordinate.

Below is our definition of Keller colorings in Lean. Notably, the `KColoring` type is a structure that includes the proofs that `C` is a valid coloring, which means that the type is empty for any pair of n and s if Keller's conjecture is true for n. We also use slightly different data types than might be expected: We use `BitVec`s rather than `Nat`s or I^n to more easily access the binary representation of each index, and we use `Vector`s instead of functions from `Fin s` to `n` to increase performance when computing with colorings.

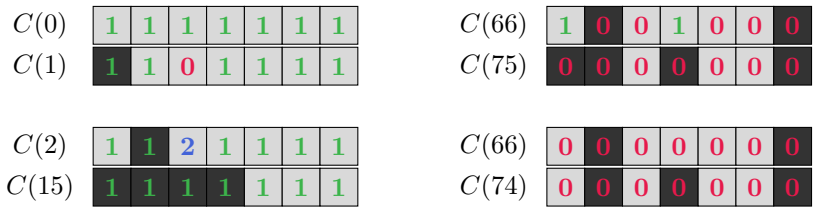


Figure 6 Examples of colorings for pairs of indices that do (left) and do not (right) respect the Keller coloring constraints. The background colors of the squares are the binary bits of the index, while the numbers are the colors assigned to each coordinate. Top left: Gap at $d = 0$, no-faceshare at $d = 2$. Bottom left: Gap at $d \in \{0, 3\}$, not adjacent. Top right: No gap. Bottom right: Gap at $d = 3$, but adjacent and lacking a no-faceshare difference.

```
structure KColoring (n s : Nat) where
  C : BitVec n → Vector (Fin s) n
  gap : ∀ (i j : BitVec n), i ≠ j → ∃ d, i[d] ≠ j[d] ∧ (C i)[d] = (C j)[d]
  no_f : ∀ (i j : BitVec n), adjacent i j → ∃ d, (C i)[d] ≠ (C j)[d]
```

Finally, to add Keller colorings to the reduction shown in Figure 3 (which would take the form of a new box on the bottom row), we proved the following theorem in Lean. It says that Keller's conjecture is true if and only if the type of Keller colorings is empty for $s = 2^{n-1}$.

```
theorem euclideanConjecture_iff_KColoring (n : Nat) :
    conjectureIn (n+1) ↔ IsEmpty (KColoring (n+1) (2^n)) := by ...
```

4 SAT Encoding of the Keller Coloring

In this section, we present our SAT encoding of the Keller coloring problem, which we verified in Lean using a project for verified SAT solving called `trestle`. Our encoding is very similar to BHMN's, although ours is more compact and allows for better propagation within the SAT solver. When combined with a better cube split (see Section 6), we were able to solve the coloring problem for $s = 2^{n-1}$ and therefore avoid formalizing the discretization results for the $s \in \{3, 4, 6\}$ cases.

4.1 Boolean Satisfiability (SAT) and `trestle`

Given a propositional formula F, the boolean satisfiability (SAT) problem asks whether there exists a truth assignment on the boolean variables of F that satisfies it. SAT solving is an entire field in its own right; we define only the relevant concepts here. For an in-depth overview, see the Handbook of Satisfiability [5], particularly Chapters 2 and 3.

Almost all modern SAT solvers require the formula F to be in *conjunctive normal form* (CNF), meaning that F is written as a conjunction of clauses $F = \bigwedge C$, where each clause is a disjunction of literals $C = \bigvee \ell$. Literals ℓ are either a boolean variable x or its negation $\overline{x}$. Let $V(F)$ and $L(F)$ be the set of variables and literals in a formula, respectively. If F is clear from context, then we will just write V and L. Unless stated otherwise, all formulas in this paper are assumed to be in CNF.

A truth assignment τ is a function $\tau : L \to \{\top, \bot\}$ respecting negation. An assignment τ satisfies a CNF formula F if it satisfies at least one literal in every clause in F. Notationally, we write $\tau \vDash \ell$ if $\tau(\ell) = \top$, $\tau \vDash C$ if $\tau \vDash \ell$ for some $\ell \in C$, and $\tau \vDash F$ if $\tau \vDash C$ for all $C \in F$. If there exists a τ that satisfies F, then F is *satisfiable*; otherwise, it is *unsatisfiable*.

One of the strengths of modern SAT solvers is that they are *certifying algorithms* [27], meaning they return answers checkable by formally-verified software. In the satisfiable case, the SAT solver returns an explicit truth assignment that can trivially be checked to satisfy all clauses in F. In the unsatisfiable case, the SAT solver emits a *proof of unsatisfiability* in a formal proof system. The de facto standard is the DRAT proof format [35]. In this paper, we use a generalization of DRAT called *substitution redundancy* (SR) [8, 11, 17] to mechanically verify symmetry-breaking clauses we add to the Keller CNF encoding (Section 5.3).

Before we can use a SAT solver, we must first *encode* our problem into a CNF formula. The choice of encoding can have a major impact on the solver's performance, with better encodings speeding up the SAT solver by up to two orders of magnitude [31]. However, encodings are often complicated and may not obviously express the original problem, which makes verifying them an important step in the SAT-solving process. Many individual SAT encodings have been formally verified [14, 32], and a few libraries exist for verifying SAT encodings in general [10, 16]. In this formalization, we use `trestle`,[5] a Lean 4 project for writing verified SAT encodings. The `trestle` project also contains a verified SR proof checker [11], which we use to check the SR proofs we generate.

Verified encodings are expressed in `trestle` using `VEncCNF`, a type that pairs an encoding `e` on a variable type ν with a proof that `e` encodes a logical *specification* P. Roughly, an encoded formula F encodes P if all truth assignments that satisfy P also satisfy F.

```
abbrev PropPred (ν : Type u) := (ν → Bool) → Prop
def VEncCNF (ν : Type u) (P : PropPred ν) :=
  { e : EncCNF ν // e.encodesProp P }     /- In Lean, '//' creates a subtype -/
```

In `trestle`, the encoding type `EncCNF` is a state monad that manages, among other things, the scoped declaration of auxiliary variables in a manner similar to quantifiers or lambda functions. For every new auxiliary variable v, the encoding reserves a fresh name for v in the CNF formula. This work is tedious and nontrivial, but the encoding monad hides much of it from the user, which streamlines the process of writing encodings.

`trestle` also provides primitives for common encoding operations. For example, the `for_all` function applies an encoding function to each element of an array and extends each sub-`VEncCNF` to one that includes the entire array:

```
def for_all (arr : Array α) {P : α → PropPred ν} (f : (a : α) → VEncCNF ν (P a))
  : VEncCNF ν (fun τ => ∀ a ∈ arr, P a τ) := ...
```

In our formalization, we make heavy use of `trestle`'s encoding primitives and `trestle`'s proof library to prove that our Keller encoding agrees with its `PropPred` specification.

4.2 Keller Specification and Encoding

We now present a CNF encoding of the Keller coloring problem, related to colorings through an intermediate logical *specification*. The specification and the encoding share the same boolean variables V and classes of constraints P. Crucially, the constraints in P are expressed in a form amenable to CNF encoding. Aside from the No-Faceshare constraints (see below), our encoding matches the one presented by BHMN in Section 4 of their paper.

[5] `https://github.com/FormalSAT/trestle`. As of this writing, there is no publication for `trestle`.

Variables. We define boolean variables of the form $x_{i,d,c}$, where $i \in I^n$ is an index expressed as a length-n binary string, $d \in \langle n \rangle$ is a coordinate, and $c \in \langle s \rangle$ is a color. Together, these variables represent the image of the Keller coloring C, where variable $x_{i,d,c}$ is true when index i in coordinate d has color c, i.e., $\tau \vDash x_{i,d,c} \iff C(i)_d = c$.

Constraints. Three types of constraints comprise P: Exactly-one, Gap, and No-Faceshare, which we label as P_1, P_2, and P_3, respectively. As their names suggest, they ensure that the Keller coloring C induced by the $\{x_{i,d,c}\}$ variables is well-formed and that the Gap and No-Faceshare coloring constraints are respected.

Exactly-one. The Exactly-one constraints P_1 ensure that every index i and every coordinate d gets assigned a unique color c. This constraint doesn't appear in Definition 1 because it is enforced by construction, but in the propositional form we must specify that exactly one of the $x_{i,d,(\cdot)}$ variables is true. Syntactically, we write $\exists!\, c,\ \tau \models x_{i,d,c}$, where in general "$\exists!\, a,\ P(a)$" means there exists a unique a such that $P(a)$ holds.

To encode P_1 into CNF, we encode "$\exists!\, c$" with at-least-one (ALO) and at-most-one (AMO) constraints. These kinds of *cardinality constraints* are well understood in the SAT literature, and many variants of AMO constraints exist [25, 29]. For our purposes, it is sufficient to use the direct (or pairwise) AMO encoding, where binary clauses prevent every pair of relevant variables from being true at the same time. Adding ALO and AMO constraints on the same set of variables $x_{i,d,(\cdot)}$ ensures that exactly one of them is true.

Formally, the specification of P_1 and its CNF encoding F_1 are:

$$P_1(\tau) \coloneqq \forall\ i\ d,\ \exists!\, c,\ \tau \models x_{i,d,c} \qquad F_1 \coloneqq \bigwedge_{i,d} \left[\left(\bigvee_{c} x_{i,d,c} \right) \wedge \bigwedge_{c<c'} (\overline{x}_{i,d,c} \vee \overline{x}_{i,d,c'}) \right].$$

Gap. In Definition 1, the Gap constraint says that all pairs of distinct indices i and j have a coordinate d where their binary strings are different but their colors $C(i)_d$ and $C(j)_d$ are the same. In terms of the x variables, this means checking that $\tau(x_{i,d,c}) = \tau(x_{j,d,c})$ for all c and some d. Formally, $P_2(\tau) \coloneqq \forall\ i \neq j,\ \exists\ d,\ i_d \neq j_d \wedge \forall\ c,\ \tau(x_{i,d,c}) = \tau(x_{j,d,c})$.

The CNF encoding of P_2 requires a bit of cleverness. Without loss of generality, let $i < j$ be two distinct indices, and let $D(i,j)$ be the set of coordinates on which i and j differ, i.e., $D(i,j) \coloneqq \{d \mid i_d \neq j_d\}$. Adopting the name used by BHMN, we introduce auxiliary variables $z_{i,j,d}$ for all $d \in D(i,j)$, where $z_{i,j,d}$ is true when i and j have the same color at coordinate d. A single clause encodes that at least one $z_{i,j,d}$ is true. To encode equality, two ternary clauses enforce that if $z_{i,j,d}$ is true, then $x_{i,d,c}$ and $x_{j,d,c}$ cannot have opposite truth values. Formally, the encoding F_2 is:

$$F_2 \coloneqq \bigwedge_{i<j} \left[\left(\bigvee_{d \in D(i,j)} z_{i,j,d} \right) \wedge \bigwedge_{d \in D(i,j),\, c} (\overline{z}_{i,j,d} \vee x_{i,d,c} \vee \overline{x}_{j,d,c}) \wedge (\overline{z}_{i,j,d} \vee \overline{x}_{i,d,c} \vee x_{j,d,c}) \right].$$

No-Faceshare. In Definition 1, the No-Faceshare constraint says that all adjacent pairs of indices i and j have pairwise unequal color vectors, meaning that $C(i)_d \neq C(j)_d$ for some coordinate d. In terms of the x variables, this means checking that $\tau(x_{i,d,c}) \neq \tau(x_{j,d,c})$ for some c and d. Formally, $P_3(\tau) \coloneqq \forall\, i\ j,\ \text{adjacent}(i,j) \implies \exists\ d\ c,\ \tau(x_{i,d,c}) \neq \tau(x_{j,d,c})$.

The CNF encoding of P_3 uses a set of auxiliary variables similar to the ones for P_2. Without loss of generality, let $i < j$ be two adjacent indices. Then let auxiliary variables $y_{i,j,d}$ be true when i and j have *different* colors in coordinate d. A single clause encodes that at least one $y_{i,j,d}$ is true. To encode inequality, we use a ternary clause to disallow the case where both of $x_{i,d,c}$ and $x_{j,d,c}$ are true. Formally, the encoding F_3 is:

$$F_3 := \bigwedge_{i<j,\,\text{adjacent}(i,j)} \left[\left(\bigvee_d y_{i,j,d} \right) \wedge \bigwedge_{d,c} (\overline{y}_{i,j,d} \vee \overline{x}_{i,d,c} \vee \overline{x}_{j,d,c}) \right].$$

Our encoding of No-Faceshare differs slightly from BHMN's. We introduce variables $y_{i,j,d}$, whereas BHMN introduce variables $y_{i,j,d,c}$ that perform a similar role. However, our set of variables has two main advantages. First, our encoding of F_3 is more compact because it uses exactly half as many clauses. Second, and more importantly, it reduces the number of auxiliary variables by a factor of s, and since $s = 64$ for the $n = 7$ case, this means quite a substantial reduction, which in turn improves propagation speed and thus performance.

Trestle representation. It is straightforward to represent the variables V and the specification P in `trestle`. Here are our definitions:

```
inductive KellerVars (n s : Nat)
  | x (i : BitVec n) (d : Fin n) (c : Fin s)


def KellerSpec {n s : Nat} : PropPred (KellerVars n s) := (fun τ =>
  /- Exactly-one constraints -/
  ( ∀ (i : BitVec n) (d : Fin n), ∃! c, τ (x i d c) ) ∧
  /- Gap constraints -/
  ( ∀ (i j : BitVec n), i ≠ j → ∃ d, i[d] ≠ i'[d]
      ∧ ∀ (c : Fin s), τ (x i d c) = τ (x j d c) ) ∧
  /- No-Faceshare constraints -/
  ( ∀ (i j : BitVec n) (d : Fin n), i ^^^ j = BitVec.oneAt d →
      ∃ d', d' ≠ d ∧ ∀ c, ¬ τ (x i d' c) ∨ ¬ τ (x j d' c) ) )
```

Expressing the constraint encodings in `trestle` is a little trickier. Whenever we declare auxiliary variables $y_{i,j,d}$ or $z_{i,j,d}$, we need to prove that they have their intended semantic meaning. Thankfully, `trestle` provides a primitive called `withTemps` that acts like a quantifier for auxiliary variables, much like `for_all`. Its definition is quite technical, but roughly, `withTemps` provides a way to prove that a specification on a base set of variables V agrees with one that includes auxiliary variables.

For example, here is the `trestle` encoding of the Gap constraint on two indices. The `(by ...)` block following the call to `mapProp` is the proof that the clauses in the encoding satisfy the specification. Thanks to `trestle`'s proof library, this proof is only 11 lines long.

```
def gapEncoding {n s} (i j : BitVec n) : VEncCNF (KellerVars n s)
    (fun τ => ∃ d, i[d] ≠ j[d] ∧ ∀ c, τ (x i d c) = τ (x j d c)) :=
( let coords := Array.finRange n
  let colors := Array.finRange s
  let D := coords.filter (fun d => i[d] ≠ j[d])
  withTemps (Fin n) <| do   /- The auxiliary variables are drawn from (Fin n) -/
    addClause (D.map (fun d => tempVar d))    /- The ALO clause -/
    for_all D fun d => do
      for_all colors fun c => do    /- The two ternary clauses -/
        addClause #[-(tempVar d),  (x i d c), -(x j d c)]
        addClause #[-(tempVar d), -(x i d c),  (x j d c)]
) |>.mapProp (by ...)  /- A proof that these clauses encode the PropPred spec -/
```

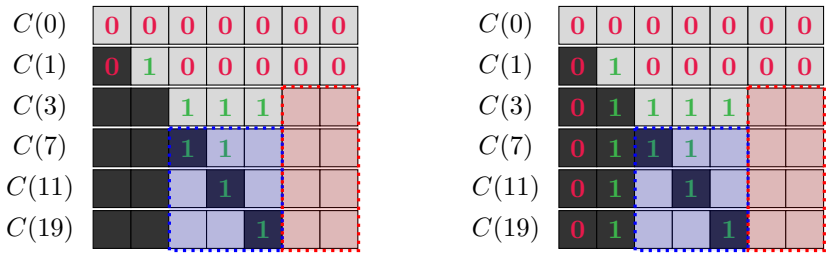


Figure 7 The symmetry-breaking constraints for $n = 7$ (left) and the units subsequently implied by unit propagation (right). The colored numbers indicate the fixed colorings for each coordinate. The blue region is the matrix symmetry block, and the red region is the dimension 5/6 block. The fixed colors in the first two rows were proved in Lean; everything else was proven using SR.

We were able to write the other two sub-encodings in a similar manner. Overall, it took only 150 lines of Lean code to write and verify the full encoding we presented above.

5 Symmetry Breaking

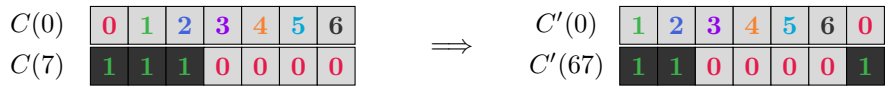
The Keller coloring problem has many interesting symmetries, and as a result, so does the encoding presented in Section 4. Unfortunately, the symmetries in the encoded formula cause the SAT solver to explore redundant areas in the search space and thus time out.

To make solving tractable, BHMN added two kinds of *symmetry-breaking constraints* to the encoded formula. First, BHMN broke symmetries on the level of the clique problem. In Section 5 of their paper, they showed that there exists a 2^n-sized clique if and only if there exists a Keller coloring with certain fixed colors for indices 0 and 1. These fixed colors were added as unit clauses to the encoding, which produced a partially-symmetry-broken propositional formula F. Next, in Section 6, BHMN broke symmetries on the propositional level by adding additional constraints to F. These constraints were mechanically justified with the DRAT proof system [35] and were checked with a formally-verified proof checker [13].

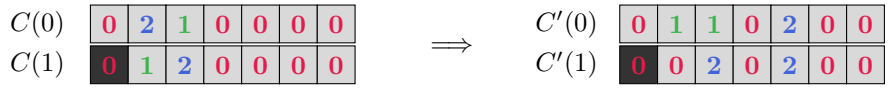
In our formalization, we verify the same kinds of symmetry constraints using similar methods. We verify the fixed colors for indices 0 and 1 in Lean by reasoning about symmetries on Keller colorings, and we express the rest of the constraints in the substitution redundancy (SR) proof system. We validate their addition to the formula with a verified SR proof checker [11]. Figure 7 summarizes the set of symmetry-breaking constraints we added.

Overall, this portion of our work left us with two lessons. The first is that long chains of symmetry reasoning are more tedious to verify in Lean than they appear on paper. This is due to how each symmetry reasoning step must be shown to preserve the constraints added by the previous steps. Future formalizations might be able to avoid this problem by developing good abstractions for the composability of different symmetries.

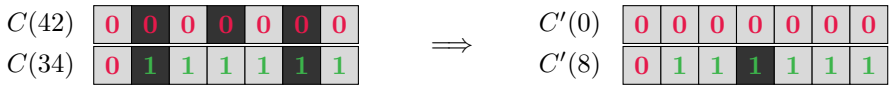
The second lesson is that, by carefully choosing the order in which to apply symmetries, it is possible to mechanically validate a significant portion of the symmetry-breaking constraints. One of the symmetries we use, the conditional index bit flip, is not expressible in SR, and so we are forced to use Lean to verify constraints that use it. But once we are done using it, we can (and do) mechanically validate all remaining clauses using SR. By moving this transition point to be as soon as possible in our proofs, we reduced the human proof burden.



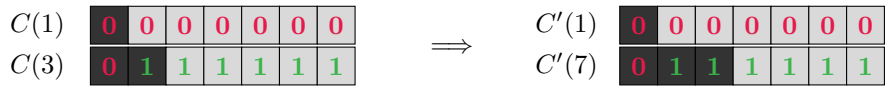
(a) Dimension permutation. All coordinates shift to the left, with the first becoming the last.



(b) Color permutation. The color permutation (2 1 0) is applied to coordinates 1 and 4.



(c) Index bit flip. The index bits are flipped in coordinates 1, 3, and 5.



(d) Conditional index bit flip. The index bit is flipped in coordinate 2 if the color is 1. Note that the CIBF may not preserve index adjacencies, but it does preserve the No-Faceshare coloring condition.

Figure 8 Examples of symmetries being applied to colorings of various indices.

5.1 Keller Coloring Symmetries

The Keller coloring problem is rich with symmetry, and many of these symmetries have direct geometric analogues. For example, flipping the dth bit in each index $i \in I^n$ corresponds to translating by $-e_d$ the indices with $i_d = 1$ (and their cubes) and all others by $+e_d$, where $\{e_0, \ldots, e_{n-1}\}$ is the standard basis for $\mathbb{R}^n$. As we will see, we can use these symmetries to construct a canonical coloring with certain fixed colors.

Our formalization uses four kinds of symmetries, illustrated in Figure 8.

a. **Coordinate permutation:** Permute the order of coordinates $d \in \langle n \rangle$ for all $i \in I^n$.
b. **Color permutation:** Permute the colors $c \in \langle s \rangle$ for a fixed coordinate d for all $i \in I^n$.
c. **Index bit-flip:** Flip the index bit in a fixed coordinate d for all $i \in I^n$.
d. **Conditional index bit-flip (CIBF):** Flip the index bit in a fixed coordinate d, but only for indices whose color in coordinate d is a given color c.

Transformations (a)-(c) are symmetries because they preserve all index adjacencies and coordinate-wise color (in)equalities. In other words, since Keller colorings are agnostic to the ordering of the coordinates, the ordering of the colors within any coordinate, and the signs of the various index bits, we are free to reorder or flip them however we wish. It is less obvious that transformation (d) is a symmetry, but it, too, is valid. Observe that a CIBF at coordinate d does not affect the value of $C(i)_d$ for any $i \in I^n$, and it only flips index bits in coordinate d. Given these facts, it is not hard to show that the relevant color (in)equalities are preserved by the CIBF, even if the relative index adjacencies get shuffled around.

In Lean, we define each symmetry as a function from Keller colorings to Keller colorings. For example, here is our definition of the index bit flip. Note that `mask` is a general bit mask used to apply potentially many bit flips at once. We use a similar trick for color permutations, where the permutation function `f : Fin n → (Fin s ≃ Fin s)` applies a color permutation `f d` to each coordinate `d`. In Lean, `^^^` is syntax for the bitwise-XOR operation.

```
def KColoring.flip (mask : BitVec n) (K : KColoring n s) : KColoring n s where
  C := fun i => K.C (i ^^^ mask)
  gap  := by intro i j ne
             have := K.gap (i ^^^ mask) (j ^^^ mask) (by simpa)
             simpa using this
  no_f := by intro i j adj
             exact K.no_f (i ^^^ mask) (j ^^^ mask) (by simpa [Keller.adjacent])
```

We also prove in Lean that each symmetry is, in fact, a symmetry. The proofs are extremely straightforward, and they amount to providing the inverse of each symmetry.

5.2 Symmetry Breaking in Lean

We now turn to verifying the first set of symmetry-breaking constraints, namely, that there exists a Keller coloring if and only if there exists a Keller coloring with colors fixed as in the first two rows of Figure 7. We additionally impose lexicographic constraints on the third row, which will help us in Section 5.3 when we add constraints using SR.

Notably, to fix the first two rows, we must assume that Keller's conjecture holds for dimension $n-1$. In our formalization this is not a problem, since we prove directly that the conjecture holds for $n = 2$, and then we incrementally prove that the conjecture holds for all $3 \le n \le 6$. Thus, we may add these symmetry-breaking constraints for any $3 \le n \le 7$.

The theorem, with a pen-and-paper proof sketch, is as follows.

▶ **Theorem 2.** *Let $n \ge 3$, and assume Keller's conjecture holds in dimension $n-1$. If there exists an n-dimensional Keller coloring C, then there exists a Keller coloring C' where $C'(0) = \vec{0}$ and $C'(1) = \big(0, 1, \cdots \vec{0} \cdots\big)$. In particular, if $n = 7$, then:*

$C'(0)$	0	0	0	0	0	0	0
$C'(1)$	0	1	0	0	0	0	0

Proof ([6], Lemma 2). Let C be a valid n-dimensional Keller coloring, and consider the $(n-1)$-dimensional coloring $\hat{C}$ formed by dropping the last coordinate from $C(i)$ for each $i < 2^{n-1}$. Let the notation $i0$ mean appending a 0 bit to the end of index i. Then formally, we have $\hat{C}(i)_d \coloneqq C(i0)_d$ for all $i \in I^{n-1}$ and $d \in \langle n-1 \rangle$.

Since we assumed that Keller's conjecture holds in dimension $n-1$, $\hat{C}$ cannot be a valid Keller coloring. However, observe that indices $i0$ and $j0$ respect the Gap condition in C. This means there exists a coordinate $d \in \langle n \rangle$ where $i0_d \neq j0_d$ and $C(i0)_d = C(j0)_d$. Clearly, $d \neq n$, so $d \in \langle n-1 \rangle$, which means i and j satisfy the Gap condition in $\hat{C}$.

Thus, $\hat{C}$ must violate the No-Faceshare condition. This means there exists a pair of adjacent indices i and j where $\hat{C}(i) = \hat{C}(j)$. But since C is a valid coloring, the No-Faceshare condition in C for $i0$ and $j0$ must be satisfied by the last coordinate, i.e., $C(i0)_n \neq C(j0)_n$.

As a result, we have found two indices $i0$ and $j0$ whose bits differ in a single coordinate $d < n$ and whose color vectors differ only at coordinate n. From here, we apply color permutations to set all relevant colors to 0 and 1, and then we apply bit flips and coordinate permutations to map indices $(i0, j0)$ to $(0, 1)$. ◀

▶ **Corollary 3.** *We may add the following unit clauses to the Keller CNF encoding:*

$$\bigwedge_d x_{0,d,0} \wedge x_{1,1,1} \wedge \bigwedge_{d \neq 1} x_{1,d,0}.$$

Next, we apply some lexicographic sorting constraints to index 3. Some helpful notation: let $[s\!:\!e] \coloneqq \{d \mid s \le d \le e\}$ be the (inclusive) set of values from start s to end e, and let $J^n \coloneqq \{i \in I^n \mid i = 2^d + 3,\, d \in [2\!:\!n]\}$ be the set of indices adjacent to 3 with an additional bit set in a single coordinate in $[2\!:\!n]$.

▶ **Theorem 4.** *Let $n \ge 3$, and let C be an n-dimensional Keller coloring with the colors fixed as in Theorem 2. Then there exists a coloring C' where $C'(3)_{[2:n]}$ is lexicographically sorted such that all positive colors appear before all 0s, and where $C'(3)_{[2:n]}$ is not lexicographically smaller than $C'(j)_{[2:n]}$ for each index $j \in J^n$ according to the ordering $0 \prec 1 = 2 = \cdots = s$. In other words, if $d \in [2\!:\!n]$ is the greatest coordinate with $C'(3)_d > 0$, then there is not an index $j \in J^n$ with $C'(j)_{d'} > 0$ for all $d' \in [2\!:\!d]$ and also $C'(j)_\delta > 0$ for some $\delta \in [d+1, n]$.*

Proof ([6], Theorem 4). Let $2 \le d \le n$ be our induction variable representing the coordinate where $C(3)$ switches from positive colors to 0s. The induction hypothesis is that $C(3)_{d'} > 0$ for every $d' \in [2\!:\!d]$. The base case of $d = n$, where index 3 only has positive colors in coordinates $[2\!:\!n]$, trivially holds.

Now assume $d < n$. We have three cases. If there is a larger coordinate $\delta > d$ with a positive color $C(3)_\delta > 0$, apply a coordinate permutation swapping coordinates $d+1$ and δ, and induct. Otherwise, there might be an index $j \in J^n$ with $C(j)_{d'} > 0$ for all $d' \in [2\!:\!d]$ (the same condition as index 3) but also with a positive color $C(3)_\delta > 0$ for some $\delta > d$. If so, apply a CIBF to swap indices 3 and j. Now we are back in the first case. If such an index doesn't exist, then the conditions of the theorem hold. ◀

Our Lean proof initially followed BHMN's pen-and-paper proof. However, their proof was very tedious to express in Lean, spanning 350 lines of uninteresting code. The tedium comes from proving that all prior constraints are preserved when five particular symmetries are applied in order. For example, we had to show that applying a color permutation that maps the positive colors $C(3)_d$ to 1 for each coordinate $d \in [2:n]$ preserves the fixed colors for indices 0 and 1. These proof obligations grew as more constraints are added.

In the end, we found a simpler proof that spans only 200 lines of less-tedious Lean code. Partly, this was due to Theorem 4 being weaker than BHMN's: the original version of the theorem fixed the colors for index 3, as shown in Figure 7. We were able to prove a weaker theorem because we discharged the rest of its proof obligations with SR.

Ideally we would have proven everything in SR, but the proof presented above uses the CIBF symmetry, which is not expressible in SR, so everything up to and including the CIBF we were forced to prove in Lean.

▶ **Corollary 5.** *We may add the following unit clauses to the Keller CNF encoding. Note that $(\overline{a} \vee b)$ is equivalent to $(a \Rightarrow b)$, and $(\bigwedge_A a) \Rightarrow (\bigwedge_B b)$ is equivalent to $\bigwedge_B (\bigvee_A \overline{a} \vee b)$.*

$$\bigwedge_{d \in [2:n)} \left[(\overline{x}_{3,d,0} \vee x_{3,d+1,0}) \wedge \bigwedge_{j \in J^n} \left[\left(\overline{x}_{3,d,0} \wedge x_{3,d+1,0} \wedge \bigwedge_{d' \in [2:d]} \overline{x}_{j,d',0} \right) \Rightarrow \left(\bigwedge_{\delta \in (d:n]} x_{j,\delta,0} \right) \right] \right]$$

We end this subsection by showing how we express these symmetry-breaking constraints in Lean. Recall from Section 3 that we related Keller's conjecture to whether the type `KColoring` is inhabited. We extend this relation by proving that, given a valid Keller coloring, we can construct a truth assignment τ that satisfies the encoded formula from Section 4:

```
def cliqueToAssn (KC : KColoring n s) : (Vars n s) → Bool :=
  fun | KellerVars.x i d c => (KC.C i)[d] = c

theorem cliqueToAssn_satisfies_KellerSpec (n s : Nat) (C : KColoring n s) :
  cliqueToAssn C |= KellerSpec := by ...
```

Next, we add the constraints from our two Theorems to `KColoring` and to `KellerSpec`. Below, `C3Zeros` extends `KColoring`, where the `extends` keyword in Lean creates a subtype of the original structure with additional fields or hypotheses. This is similar to inheritance in many standard programming languages.

```
def fullSpec : PropPred (Vars n s) :=
  fun τ => KellerSpec τ ∧ c0_c1_spec τ ∧ c3_sorted_spec τ

theorem cliqueToAssn_satisfies_fullSpec (n s : Nat) (C : C3Zeros n s) :
  cliqueToAssn C.toKColoring |= fullSpec := by ...
```

Finally, we add the CNF version of the symmetry-breaking constraints to our encoding. Since we use `VEncCNF`, we also provide a proof that this encoding agrees with `fullSpec`.

```
def fullEncoding (n s : Nat) : VEncCNF (Vars n s) fullSpec :=
( KellerEncoding n s            /- The encoding from Section 4 -/
  ∧ guard (n ≥ 2 ∧ s ≥ 2)      /- Only adds clauses if the guard holds -/
    ( c0_c1 (n-2) (s-2) ∧ c3_sorted (n-2) (s-2) )
) |>.mapProp ( by ... )   /- A proof that the clauses encode fullSpec -/
```

Our encoding can be compiled, and running the compiled binary writes the encoded formula to disk. In principle, this formula could then be given to a SAT solver, but it would still be intractable to solve. As a result, we must break more symmetries, this time with SR.

5.3 Symmetry Breaking In SR

The next step is to break symmetries at the propositional level by adding clauses to the partially-symmetry-broken CNF formula. These new clauses are not verified in Lean directly; rather, we write the clauses to a separate file written in the *substitution redundancy* (SR) proof system [8, 11, 17]. A formally-verified SR proof checker [11] then checks their validity, and the combined formula is what we ultimately give to the SAT solver.

Even though these SR clauses were verified mechanically, we found them manually. We were guided by a similar kind of symmetry breaking done by BHMN in Section 6 of their paper, but since our encoding is slightly different from theirs, we had to experiment with different sets of SR clauses to find the one that would enable effective SAT solving. In principle, we could have used automated tools such as SHATTER [1] or the state-of-the-art SR-compatible symmetry-breaking tool SATSUMA [2], but manual methods gave us the control we needed to ensure efficient solving.

After briefly introducing SR, we spend the rest of this section discussing the different kinds of SR clauses we add to the formula. As part of our formalization, we wrote a short program in Lean that generates the SR files for these clauses.

Substitution redundancy is a *clausal proof system* that generalizes the popular RAT proof system [35]. Each step of a clausal proof either adds or deletes a clause from a CNF formula F. To add a clause C, the proof checker must check that C is *redundant*, meaning that F and $F \wedge C$ are *equisatisfiable*. Here, it is sufficient to show that if F is satisfiable, so is $F \wedge C$. A proof that adds the empty clause $\bot$ is a *proof of unsatisfiability* for F, while a proof that only adds clauses is a *proof of symmetry breaking*.

When adding a clause C, the proof step may optionally include a *witness* σ that helps show that C is redundant. In SR, the witness is a *substitution* $\sigma : V(F) \to \{\top, \bot\} \cup L(F)$ that maps the formula's variables to fixed truth values or to other formula literals. We apply σ to a formula F, written as $F|_\sigma$, by mapping the variables in F under σ.

In SR, the eponymous SR rule implies redundancy [11]. Syntactically, the SR rule is written as $F \wedge \neg C \vdash_1 (F \wedge C)|_\sigma$, where $\vdash_1$ denotes entailment under unit propagation. Intuitively, if an assignment τ satisfies F but not C, then σ "repairs" τ into an assignment satisfying both.

Many clausal proof systems, including RAT and SR, have *unhinted* (DRAT, DSR) and *hinted* (LRAT, LSR) proof formats, the main difference being that the hinted versions include unit propagation hints to make proof checking take time linear in the size of the proof. Verified proof checkers for these proof systems accept only the hinted versions.

In this project, we use DSR-TRIM,[6] an unverified DSR proof checker written in C that converts DSR proofs into LSR proofs. We then use the verified LSR proof checker [11] implemented in `trestle` to check the hinted proofs.

Luckily for us, SR substitutions σ can naturally express the Keller coloring symmetries from Section 5.1. For example, suppose we want to add the unit clause $(\overline{x}_{3,2,s})$ to ensure that $C(3)_2 \neq s$, with $s \neq 1$. If we ever have a truth assignment that satisfies the encoded formula F but sets $x_{3,2,s}$ to true, then we may apply a color permutation swapping colors s and 1 in coordinate 2. The SR substitution would then be $\sigma(x) \coloneqq (x_{3,2,s} \mapsto x_{3,2,1}, x_{3,2,1} \mapsto x_{3,2,s})$, where any omitted variables are mapped to themselves. In actuality, the substitutions are more complicated than this, since we also need to remap any affected auxiliary variables $y_{3,j,2}$ and $z_{3,j,2}$, but we handle these details in our formalization.

For the $n = 7$ case, we generate five kinds of SR clauses. We briefly describe each.

Index color bounds. Observe that for any list of indices $L = [i_1, \ldots, i_k]$, there are at most k distinct colors in any coordinate d. Thus, we may apply color permutations to bound the colors for each index by its position in L, i.e., $C(i_1)_d \leq 0$, $C(i_2)_d \leq 1$, and so on.

We apply a similar technique to the indices 3, 7, 11, and 19 for coordinates $[2\colon4]$, noting that we have already fixed the color 0 for indexes 0 and 1 as in Figure 7. Thus, $C(3)_d \leq 1$, $C(7)_d \leq 2$, and so on. To accomplish this, we add the unit literals $(\overline{x}_{i,d,c})$ for $d \in [2\colon4]$ and for c greater than the desired bound. The SR witness is the appropriate color permutation.

Index 3 fixing. After bounding the colors for index 3, a handful of clauses become implied (RUP) by the formula. In particular, we may fix $C(3)_d = 1$ for $d \in [2\colon4]$.

Increment sorting. Observe that for any list of indices $L = [i_1, \ldots, i_k]$ that have been color bounded, we can further enforce that the colors along any coordinate d be *increment sorted*, meaning that each successive color is no more than one higher than the previous maximum. We add clauses that block solutions that are not increment sorted for indices 7, 11, and 19 among the coordinates in $[2\colon5]$. The SR witnesses are color permutations.

Matrix symmetry. At this point, the 1s along the diagonal of the blue matrix are fixed, as shown in Figure 7. However, there are still matrix symmetries among the color assignments to the remaining matrix entries. As discussed in Section 6.1 of BHMN's paper, for $s \geq 4$ there are 28 canonical assignments to the matrix, and for each assignment, we reorder the columns of the matrix to find the lexicographically smallest matrix coloring. We add clauses to block the non-canonical assignments.

[6] `https://github.com/ccodel/dsr-trim`

Coordinates 5 and 6. For $n = 7$, coordinates 5 and 6 are potentially still symmetric. We already lexicographically sorted index 3, so $C(3)_5 \succeq C(3)_6$. However, if these colors are equal (meaning they are both 0 or both 1), then we can swap coordinates 5 and 6 to enforce a lexicographically larger coloring in coordinate 5 than in coordinate 6 among the indices 7, 11, and 19. In other words, we can treat the four relevant colors in each coordinate as a string (read top-down in the figure), and swap if needed to get the larger string in coordinate 5.

6 Solving the CNFs

Finally, we solve the encoded CNF formulas with a SAT solver. To do so, we must navigate a few challenges. First, recall that the symmetry breaking for dimension n relies on Keller's conjecture holding in dimension $n - 1$. We handle this in Lean by proving a sequence of results from lower dimensions up to $n = 7$. Second, a significant portion of the symmetry breaking is justified with SR clauses that must be validated. Third, the formula for $n = 7$ is too hard for conventional SAT solvers, even with the improved encoding and effective symmetry breaking. To address this issue, we use a common SAT-solving technique called *cube and conquer* [19], as was done in the original BHMN paper.

Starting with $n = 2$, we solve the base encoding without symmetry breaking using a conventional single-threaded CDCL solver CaDiCaL [4], which outputs an LRAT proof. This proof is checked by a verified LRAT checker [34], and then a proof-by-reflection tool gives a complete proof in Lean that the conjecture holds for $n = 2$. We repeat this process for $3 \leq n \leq 5$, now including the symmetry-breaking clauses from Section 5.2 that are justified by the result for the previous dimension. These results are all verified within a few seconds.

However, the situation is more challenging for $n = 6, 7$. In both cases, the formulas only become tractable after adding the SR symmetry-breaking clauses, so we use a verified tool to check that the SR proofs are correct. For $n = 6$ this check takes a few minutes, while for $n = 7$ it takes just over an hour. After adding the SR clauses, the $n = 6$ formula can be solved quickly by CaDiCaL. But this is still insufficient for $n = 7$. We discuss this below.

6.1 Cube and Conquer

Cube and conquer is a technique for solving hard CNFs. A formula F is broken into many subformulas $F \wedge \alpha_i$, where the α_i are conjunctions of literals (called "cubes") used to guide the solver [19]. As long as the cubes together form a tautology, meaning they cover the search space, solving the many subformulas is equivalent to solving F. The original BHMN result splits on all 28 ways of assigning the blue "matrix block" and all 1378 ways of assigning the red "dimension 5/6 columns" from Figure 7. This gave a total of 38,584 cubes to check.

We used proofix [3], a new tool for automatically identifying good variables to split on, to investigate ways to improve this split. Across many runs, the tool would consistently split on just one of the variables in the matrix block. We determined that the tool was splitting between the *hardest* matrix assignment and the 27 other matrix assignments. We implemented this insight, while still splitting all 1378 cases for the red columns. As with BHMN, one cube proved substantially harder than the others. We split this cube into 16 subcubes using all possible assignments to four variables suggested by proofix ($x_{15,5,0}$, $x_{22,0,0}$, $x_{23,6,0}$, $x_{26,0,0}$). Table 1 shows the size of the formulas and the proofs, and the time to produce and validate the proofs. Compared to BHMN, our improved encoding roughly halved the solving time for $s = 6$. On subproblems, we observed that the performance improvement on $s = 64$ was more substantial. After the subformulas were all solved, we checked that the formula with the negated cubes is easy to refute, ensuring that the cubes cover the entire search space.

Table 1 Statistics on solving the harder formulas. The first four columns show the dimension (n), the number of colors (s), and the number of clauses and variables of the verified encoding. The next three columns show the number of SR clauses, the time it took to add hints, and the time to validate the hinted proof (in seconds). The last three columns show the number of cubes, the total time to solve the formula, and the total time to check the proof produced by the solver (in hours).

n	s	# Cls	# Var	# SR	dsr-trim (s)	lsr-check (s)	# Cube	Solve (h)	Check (h)
6	32	617,017	25,536	1019	74.374	14.776	104	0.014	0.023
7	2	130,470	61,824	35	5.246	1.030	287	1.366	1.119
7	6	383,142	65,408	385	72.939	10.610	2771	29.332	19.578
7	64	5,657,894	117,376	2582	3991.633	370.542	2771	228.517	124.996

6.2 Higher Dimensions

For dimensions $n \geq 8$, Keller's conjecture fails. Clune's formalization proves that the conjecture fails for $n = 8$ by computing an explicit counterexample. This counterexample served as additional evidence that the conjecture was defined correctly in the theorem prover. However, Lean 3, the tool used in that formalization, had poor support for proof by reflection, and the $n = 8$ result took hours to check.

We give a formal proof that Keller's conjecture fails for $n \geq 8$. We use Mackey's counterexample for $n = 8$ [24], and we formally verify a program that checks that the counterexample is valid. Again using proof by reflection, we obtain a fully-verified proof that the conjecture fails for $n = 8$, and thanks to improvements between Lean 3 and Lean 4, this proof can be checked in seconds. We also prove that a counterexample in dimension n implies a counterexample in dimension $n + 1$, which is nontrivial but easy to show.

Thus, we formally resolve Keller's conjecture: it holds for $n \leq 7$ and fails for $n \geq 8$.

7 Conclusion

Although we resolved Keller's conjecture in particular, our formalization covers many standard techniques used to resolve open mathematical questions with SAT solvers. We hope that our efforts inspire further development of proof automation and tools for writing encodings, breaking symmetries, and validating SAT proofs.

References

1 F.A. Aloul, I.L. Markov, and K.A. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 836–839, 2003. doi:10.1145/775832.776042.

2 Markus Anders, Cayden R. Codel, and Marijn J. H. Heule. Orbitopal fixing in SAT. In Sebastian Junges and Guy Katz, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 32nd International Conference, TACAS 2026, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2026, Turin, Italy, April 11-16, 2026, Proceedings, Part I*, Lecture Notes in Computer Science, pages 89–109. Springer, 2026. doi:10.1007/978-3-032-22752-2_5.

3 Zachary Battleman, Joseph E. Reeves, and Marijn J. H. Heule. Problem Partitioning via Proof Prefixes. In Jeremias Berg and Jakob Nordström, editors, *28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025)*, volume 341 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAT.2025.3.

4 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

5 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.

6 Joshua Brakensiek, Marijn Heule, John Mackey, and David Narváez. The Resolution of Keller's Conjecture. *Journal of Automated Reasoning*, 66(3):277–300, 2022. doi:10.1007/s10817-022-09623-5.

7 Curtis Bright, Kevin K. H. Cheung, Brett Stevens, Ilias Kotsireas, and Vijay Ganesh. A SAT-based resolution of Lam's Problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3669–3676, May 2021. doi:10.1609/aaai.v35i5.16483.

8 Sam Buss and Neil Thapen. DRAT and propagation redundancy proofs without new variables. *Logical Methods in Computer Science*, Volume 17, Issue 2, April 2021. doi:10.23638/LMCS-17(2:12)2021.

9 Joshua Clune. A Formalized Reduction of Keller's Conjecture. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, pages 90–101, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3573105.3575669.

10 Cayden R. Codel, Jeremy Avigad, and Marijn J. H. Heule. Verified Encodings for SAT Solvers. In *2023 Formal Methods in Computer-Aided Design (FMCAD)*, pages 141–151, 2023. ISSN: 2708-7824. doi:10.34727/2023/isbn.978-3-85448-060-0_22.

11 Cayden R. Codel, Marijn J. H. Heule, and Jeremy Avigad. Verified substitution redundancy checking. In Nina Narodytska and Philipp Rümmer, editors, *Proceedings of the 24th Conference on Formal Methods in Computer-Aided Design - FMCAD 2024*, volume 5 of *Formal Methods in Computer-Aided Design*, pages 186–196, Vienna, Austria, October 2024. TU Wien Academic Press. doi:10.34727/2024/isbn.978-3-85448-065-5_24.

12 K. Corrádi and S. Szabó. A combinatorial approach for Keller's conjecture. *Periodica Mathematica Hungarica*, 21(2):95–100, 1990. doi:10.1007/BF01946848.

13 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo De Moura, editor, *Automated Deduction – CADE 26*, volume 10395, pages 220–236. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-63046-5_14.

14 Luís Cruz-Filipe, João Marques-Silva, and Peter Schneider-Kamp. Formally verifying the solution to the Boolean Pythagorean triples problem. *J. Autom. Reason.*, 63(3):695–722, 2019. doi:10.1007/S10817-018-9490-4.

15 Jennifer Debroni, John D. Eblen, Michael A. Langston, Wendy Myrvold, Peter Shor, and Dinesh Weerapurage. A complete resolution of the Keller maximum clique problem. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 129–135. Society for Industrial and Applied Mathematics, January 2011. doi:10.1137/1.9781611973082.11.

16 Sofia Giljegård and Johan Wennerbreck. Puzzle solving with proof. Master's thesis, Chalmers University of Technology, University of Gothenburg, 2021.

17 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-boolean proofs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3768–3777. AAAI Press, 2021. doi:10.1609/AAAI.V35I5.16494.

18 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-40970-2_15.

19 Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

20 Marijn J. H. Heule and Manfred Scheucher. Happy ending: An empty hexagon in every set of 30 points. In *Tools and Algorithms for the Construction and Analysis of Systems: 30th International Conference, TACAS 2024*, pages 61–80, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-57246-3_5.

21 Andrzej P. Kisielewicz. Rigid polyboxes and Keller's conjecture, 2014. arXiv:1304.1639 [math]. doi:10.48550/arXiv.1304.1639.

22 Andrzej P. Kisielewicz and Magdalena Łysakowska. On Keller's Conjecture in Dimension Seven. *The Electronic Journal of Combinatorics*, 22(1):P1.16, 2015. doi:10.37236/4153.

23 Jeffrey C. Lagarias and Peter W. Shor. Keller's cube-tiling conjecture is false in high dimensions. *Bull. Amer. Math. Soc. (N.S.)*, 1992. Publisher: arXiv Version Number: 1. doi:10.48550/arXiv.MATH/9210222.

24 Mackey. A Cube Tiling of Dimension Eight with No Facesharing. *Discrete & Computational Geometry*, 28(2):275–279, August 2002. doi:10.1007/s00454-002-2801-9.

25 Joao Marques-Silva and Inês Lynce. Towards robust CNF encodings of cardinality constraints. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 483–497, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

26 The Mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, POPL '20, January 2020. doi:10.1145/3372885.3373824.

27 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. doi:10.1016/J.COSREV.2010.09.009.

28 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pages 625–635, 2021. doi:10.1007/978-3-030-79876-5_37.

29 Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, and Pedro Barahona. Empirical study on SAT-encodings of the at-most-one constraint. In *The 9th International Conference on Smart Media and Applications*, SMA 2020, pages 470–475, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3426020.3426170.

30 Oskar Perron. Über lückenlose Ausfüllung desn-dimensionalen Raumes durch kongruente Würfel. *Mathematische Zeitschrift*, 46(1):1–26, December 1940. doi:10.1007/BF01181421.

31 Bernardo Subercaseaux and Marijn J. H. Heule. The packing chromatic number of the infinite square grid is 15. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 389–406, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-30823-9_20.

32 Bernardo Subercaseaux, Wojciech Nawrocki, James Gallicchio, Cayden Codel, Mario Carneiro, and Marijn J. H. Heule. Formal Verification of the Empty Hexagon Number. In *15th International Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:19, 2024. doi:10.4230/LIPIcs.ITP.2024.35.

33 S. Szabó. A reduction of Keller's conjecture. *Periodica Mathematica Hungarica*, 17(4):265–277, December 1986. doi:10.1007/BF01848388.

34 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. Verified propagation redundancy and compositional unsat checking in cakeml. *International Journal on Software Tools for Technology Transfer*, 25(2):167–184, 2023. doi:10.1007/s10009-022-00690-y.

35 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014*, volume 8561, pages 422–429. Springer International Publishing, 2014. doi:10.1007/978-3-319-09284-3_31.

36 Magdalena Łysakowska. Extended Keller graph and its properties. *Quaestiones Mathematicae*, 42(4):551–560, April 2019. doi:10.2989/16073606.2018.1462865.