Forum of Mathematics, Pi (2017), Vol. 5, e2, 29 pages
doi:10.1017/fmp.2017.1

A FORMAL PROOF OF THE KEPLER CONJECTURE

THOMAS HALES[1], MARK ADAMS[2,3], GERTRUD BAUER[4], TAT DAT DANG[5], JOHN HARRISON[6], LE TRUONG HOANG[7], CEZARY KALISZYK[8], VICTOR MAGRON[9], SEAN MCLAUGHLIN[10], TAT THANG NGUYEN[7], QUANG TRUONG NGUYEN[1], TOBIAS NIPKOW[11], STEVEN OBUA[12], JOSEPH PLESO[13], JASON RUTE[14], ALEXEY SOLOVYEV[15], THI HOAI AN TA[7], NAM TRUNG TRAN[7], THI DIEP TRIEU[16], JOSEF URBAN[17], KY VU[18] and ROLAND ZUMKELLER[19]

[1] University of Pittsburgh, USA;
email: hales@pitt.edu, nguyenquangtruong270983@gmail.com
[2] Proof Technologies Ltd, UK
[3] Radboud University, Nijmegen, The Netherlands;
email: mark@proof-technologies.com
[4] ESG – Elektroniksystem- und Logistik-GmbH, Germany;
email: Gertrud.Bauer@alumni.tum.de
[5] CanberraWeb, 5/47-49 Vicars St, Mitchell ACT 2911, Australia;
email: dangtatdatusb@gmail.com
[6] Intel Corporation, USA;
email: johnh@ecsmtp.pdx.intel.com
[7] Institute of Mathematics, Vietnam Academy of Science and Technology, Vietnam;
email: hltruong@math.ac.vn, ntthang.math@gmail.com, tthan@math.ac.vn, tntrung@math.ac.vn
[8] University of Innsbruck, Austria;
email: cezary.kaliszyk@uibk.ac.at
[9] CNRS VERIMAG, France;
email: magron@lix.polytechnique.fr
[10] Amazon, USA;
email: seanmcl@gmail.com
[11] Technische Universität München, Germany;
email: nipkow@in.tum.de
[12] University of Edinburgh, UK;
email: sobua@inf.ed.ac.uk
[13] Philips Electronics North America Corporation – Andover, MA, USA;
email: joe.pleso@gmail.com
[14] The Pennsylvania State University, USA;
email: jason.rute@gmail.com

[15] University of Utah, USA;
email: solovyev.alexey@gmail.com
[16] AXA China Region Insurance Company Limited, Hong Kong;
email: trieudiep87@gmail.com
[17] Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), Czech Republic;
email: urban@cs.ru.nl
[18] Chinese University of Hong Kong, Hong Kong;
email: vukhacky@gmail.com
[19] email: Roland.Zumkeller@gmail.com

Received 21 November 2014; accepted 9 December 2016

Abstract

This article describes a formal proof of the Kepler conjecture on dense sphere packings in a combination of the HOL Light and Isabelle proof assistants. This paper constitutes the official published account of the now completed Flyspeck project.

2010 Mathematics Subject Classification: 52C17

1. Introduction

The booklet *Six-Cornered Snowflake*, which was written by Kepler in 1611, contains the statement of what is now known as the Kepler conjecture: no packing of congruent balls in Euclidean three-space has density greater than that of the face-centered cubic packing [27]. This conjecture is the oldest problem in discrete geometry. The Kepler conjecture forms part of Hilbert's 18th problem, which raises questions about space groups, anisohedral tilings, and packings in Euclidean space. Hilbert's questions about space groups and anisohedral tiles were answered by Bieberbach in 1912 and Reinhardt in 1928. Starting in the 1950s, Fejes Tóth gave a coherent proof strategy for the Kepler conjecture and eventually suggested that computers might be used to study the problem [6]. The truth of the Kepler conjecture was established by Ferguson and Hales in 1998, but their proof was not published in full until 2006 [18].

The delay in publication was caused by the difficulties that the referees had in verifying a complex computer proof. Lagarias has described the review process [30]. He writes, 'The nature of this proof ... makes it hard for humans to check every step reliably. ... [D]etailed checking of many specific assertions found them to be essentially correct in every case. The result of the reviewing process produced in these reviewers a strong degree of conviction of the essential correctness of this proof approach, and that the reduction method led to nonlinear programming problems of tractable size.' In the end, the proof was published without complete certification from the referees.

At the Joint Math Meetings in Baltimore in January 2003, Hales announced a project to give a formal proof of the Kepler conjecture and later published a project description [15]. The project is called *Flyspeck*, an expansion of the acronym FPK, for the Formal Proof of the Kepler conjecture. The project has formalized both the traditional text parts of the proof and the parts of the proof that are implemented in computer code as calculations. The project was officially completed on August 10, 2014. This paper constitutes the official published account of the Flyspeck project.

The first definite contribution to the project was the formal verification by Bauer and Nipkow of a major piece of computer code that was used in the proof (see Section 7). Major work on the text formalization project started with NSF funding in 2008. An international conference on formal proofs and the Flyspeck project at the Institute of Math (VAST) in Hanoi in 2009 transformed the project into a large international collaboration. The PhD theses of Bauer [4], Obua [39], Zumkeller [48], Magron [32], and Solovyev [43] have been directly related to this project. The book 'Dense Sphere Packings' gives the mathematical details of the proof that was formalized [17]. This article focuses on the components of the formalization project, rather than the mathematical details of the proof.

The Flyspeck project has roughly similar size and complexity as other major formalization projects such as the Feit–Thompson odd order theorem [9], the CompCert project giving a verified C compiler [31], and the seL4 project that verified an operating system microkernel [28]. The Flyspeck project might actually set the current record in terms of lines of code in a verification project. Unlike these other projects, our final result has the weakness of being spread over two different proof assistants and multiple computers in a cloud computation.

The code and documentation for the Flyspeck project are available at a github code repository devoted to the project [7]. The parts of the project that have been carried out in Isabelle are available from the Archive of Formal Proofs (afp.sf.net). Other required software tools are Isabelle/HOL [37], HOL Light [23], OCaml (the implementation language of HOL Light), the CamlP5 preprocessor (for a syntax extension to OCaml for parsing of mathematical terms), and GLPK (for linear programming) [8].

The main statement in the Kepler conjecture can be formally verified in about five hours on a 2 GHz CPU directly from the proof scripts. As described in Section 4.5, the proof scripts of the main statement have been saved in a recorded proof format. In this format, the formal proof of the main statement executes in about forty minutes on a 2 GHz CPU. We encourage readers of this article to make an independent verification of this main statement on their computers.

This main statement reduces the Kepler conjecture to three computationally intensive subclaims that are described in Section 3. Two of these subclaims

can be checked in less than one day each. The third and most difficult of these subclaims takes about 5000 CPU hours to verify.

2. The HOL Light proof assistant

The formal verifications have been carried out in the HOL Light and Isabelle proof assistants, with a second verification of the main statement in HOL Zero.

HOL Light is one of a family of proof assistants that implement the HOL logic [11]. It is a classical logic based on Church's typed lambda calculus and has a simple polymorphic type system. For an overview of the deductive system see [22]. There are ten primitive inference rules and three mathematical axioms: the axiom of infinity (positing the existence of a function $f : X \to X$ that is one to one but not onto), the axiom of extensionality (which states that two functions are equal if their outputs agree on every input), and the axiom of choice. HOL Light has a mechanism that permits the user to extend the system with new constants and new types.

HOL Light is implemented according to a robust software architecture known as the 'LCF approach' [10], which uses the type system of its implementation language (OCaml in the case of HOL Light) to ensure that all deduction must ultimately be performed by the logic's primitive deductive system implemented in a kernel. This design reduces the risk of logical unsoundness to the risk of an error in the kernel (assuming the correct implementation of the architecture).

The kernel of HOL Light is remarkably small, amounting to just a few hundred lines of code. Furthermore, the kernel has been subject to a high degree of scrutiny to guarantee that the underlying logic is consistent and that its implementation in code is free of bugs [13, 29]. This includes a formal verification of its own correctness [20]. It is generally held by experts that it is extremely unlikely for the HOL Light proof assistant to create a 'false' theorem, except in unusual circumstances such as a user who intentionally hacks the system with malicious intent. A formal proof in the HOL Light system is more reliable by orders of magnitude than proofs published through the traditional process of peer review.

One feature that makes HOL Light particularly suitable for the Flyspeck project is its large libraries of formal results for real and complex analysis. Libraries include multivariate (gauge) integration, differential calculus, transcendental functions, and point-set topology on $\mathbb{R}^n$.

Proof scripts in HOL Light are written directly as OCaml code and are executed in the OCaml read–eval–print loop. Mathematical terms are expressed in a special syntax, set apart from ordinary OCaml code by backquotes. For example, typed at the OCaml prompt, `` `1` `` equals the successor of zero in the

set of natural numbers (defined from the mathematical axiom of infinity in HOL Light). It is not to be conflated with the informal 1 in OCaml, which is vulnerable to the usual properties of machine arithmetic.

This article displays various terms that are represented in HOL Light syntax. For the convenience of the reader, we note some of the syntactic conventions of HOL Light. The syntax is expressed entirely with ASCII characters. In particular, the universal quantifier (∀) is written as (!), the existential quantifier (∃) is written (?), the embedding of natural numbers into the real numbers is denoted (&), so that for instance `&3` denotes the real number 3.0. The symbol `|-` (the keyboard approximation for ⊢) appears in front of a statement that is a theorem in the HOL Light system. Other logical symbols are given by keyboard approximations: `==>` for ⟹, `/\` for ∧, and so forth.

The logic of Isabelle/HOL is similar to that of HOL Light, but it also includes a number of features not available in HOL Light. The logic supports a module system and type classes. The package includes extensive front-end support for the user and an intuitive proof scripting language. Isabelle/HOL supports a form of computational reflection (which is used in the Flyspeck project) that allows executable terms to be exported as ML and executed, with the results of the computation re-integrated in the proof assistant as theorems.

3. The statement

As mentioned in the introduction, the Kepler conjecture asserts that no packing of congruent balls in Euclidean three-space can have density exceeding that of the face-centered cubic packing. That density is $\pi/\sqrt{18}$, or approximately 0.74. The hexagonal-close packing and various packings combining layers from the hexagonal-close packing and the face-centered cubic packing also achieve this bound. The theorem that has been formalized does not make any uniqueness claims.

The density of a packing is defined as a limit of the density obtained within finite containers, as the size of the container tends to infinity. To make the statement in the formal proof as simple as possible, we formalize a statement about the density of a packing inside a finite spherical container. This statement contains an error term. The ratio of the error term to the volume of the container tends to zero as the volume of the container tends to infinity. Thus in the limit, we obtain the Kepler conjecture in its traditional form.

As a ratio of volumes, the density of a packing is scale invariant. There is no loss of generality in assuming that the balls in the packing are normalized to have unit radius. We identify a packing of balls in $\mathbb{R}^3$ with the set V of centers of the balls, so that the distance between distinct elements of V is at

least 2, the diameter of a ball. More formally, we have the following theorem that characterizes a packing.

```
|- packing V <=>
     (!u v. u IN V /\ v IN V /\ dist(u,v) < &2 ==> u = v)
```

This states that $V \subset \mathbb{R}^3$ is a packing if and only if for every $\mathbf{u}, \mathbf{v} \in V$, if the distance from $\mathbf{u}$ to $\mathbf{v}$ is less than 2, then $\mathbf{u} = \mathbf{v}$. To fix the meaning of what is to be formalized, we define the constant `the_kepler_conjecture` as follows:

```
|- the_kepler_conjecture <=>
     (!V. packing V
       ==> (?c. !r. &1 <= r
            ==> &(CARD(V INTER ball(vec 0,r))) <=
                pi * r pow 3 / sqrt(&18) + c * r pow 2))
```

In words, we define the Kepler conjecture to be the following claim: for every packing V, there exists a real number c such that for every real number $r \geqslant 1$, the number of elements of V contained in an open spherical container of radius r centered at the origin is at most

$$\frac{\pi\, r^3}{\sqrt{18}} + c\, r^2.$$

An analysis of the proof shows that there exists a small computable constant c that works uniformly for all packings V, but we only formalize the weaker statement that allows c to depend on V. The restriction $r \geqslant 1$, which bounds r away from 0, is needed because there can be arbitrarily small containers whose intersection with V is nonempty.

The proof of the Kepler conjecture relies on a combination of traditional mathematical argument and three separate bodies of computer calculations. The results of the computer calculations have been expressed in precise mathematical terms and specified formally in HOL Light. The computer calculations are as follows.

(1) The proof of the Kepler conjecture relies on nearly a thousand nonlinear inequalities. The term `the_nonlinear_inequalities` in HOL Light is the conjunction of these nonlinear inequalities. See Section 5.

(2) The combinatorial structure of each possible counterexample to the Kepler conjecture is encoded as a plane graph satisfying a number of restrictive conditions. Any graph satisfying these conditions is said to be *tame*. A list

of all tame plane graphs up to isomorphism has been generated by an exhaustive computer search. The formal statement that every tame plane graph is isomorphic to one of these cases can be expressed in HOL Light as `import_tame_classification`. See Section 7.

(3) The final body of computer code is a large collection of linear programs, formally specified as `linear_programming_results` in HOL Light. See Section 9.

It is then natural to break the formal proof of the Kepler conjecture into four parts: the formalization of the text part (that is, the traditional noncomputer portions of the proof), and the three separate bodies of computer calculations. Because of the size of the formal proof, the full proof of the Kepler conjecture has not been obtained in a single session of HOL Light. What we formalize in a single session is a theorem

```
|-  the_nonlinear_inequalities /\
    import_tame_classification
    ==> the_kepler_conjecture
```

This theorem represents the formalization of two of the four parts of the proof: the text part of the proof and the linear programming. It leaves the other two parts (nonlinear inequalities and tame classification) as explicit assumptions. The formal proof of the assumption `the_nonlinear_inequalities` is described in Section 5. The formal proof of `import_tame_classification` in Isabelle is described in Section 7. Thus, combining all these results from various sessions of HOL Light and Isabelle, we have obtained a formalization of every part of the proof of the Kepler conjecture.

4. Text formalization

The next sections of this article turn to each of the four parts of the proof of the Kepler conjecture, starting with the text part of the formalization in this section. In the remainder of this article, we will call the proof of the Kepler conjecture as it appears in [**18**] the *original proof*. The proof that appears in [**17**] will be called the *blueprint proof*. The formalization closely follows the blueprint proof. In fact, the blueprint and the formalization were developed together, with numerous revisions of the text in response to issues that arose in the formalization.

Since the blueprint and formal proof were developed at the same time, the numbering of lemmas and theorems continued to change as project took shape. The blueprint and formal proof are cross-linked by a system of identifiers that persist through project revisions, as described in [**47**, Section 5].

4.1. Standard libraries. The Flyspeck proof rests on a significant body of mathematical prerequisites, including basics of measure, geometric notions, and properties of polyhedra and other affine and convex sets in $\mathbb{R}^3$. HOL Light's standard library, largely under the influence of Flyspeck, has grown to include a fairly extensive theory of topology, geometry, convexity, measure and integration in $\mathbb{R}^n$. Among the pre-proved theorems, for example, are the Brouwer fixed-point theorem, the Krein–Milman theorem and the Stone–Weierstrass theorem, as well as a panoply of more forgettable but practically indispensable lemmas.

Besides the pre-proved theorems, the library includes a number of automated procedures that can make formal proofs much less painful. In particular, one tool supports the common style of picking convenient coordinate axes 'without loss of generality' by exploiting translation, scaling and orthogonal transformation. It works by automatically using a database of theorems asserting invariance of various properties under such transformations [21]. This is invaluable for more intricate results in geometry, where a convenient choice of frame can make the eventual algebraic form of a geometrical problem dramatically easier.

The text formalization also includes more specialized results such as measures of various shapes that are particularly significant in the partitioning of space in the Flyspeck proof. Among these, for example, is the usual 'angle sum' formula for the area of a spherical triangle, or more precisely, the volume of the intersection of its conic hull and a ball. Flyspeck also uses results about special combinatorial and geometrical objects such as *hypermaps* and *fans*, and a substantial number of nontrivial results are proved about these, including in effect the Euler characteristic formula for planar graphs.

4.2. Blueprint proof outline. It is not our purpose to go into the details of the proof of the Kepler conjecture, for which we refer the reader to [17]. We simply recall enough of the high-level strategy to permit an understanding of the structure of the formalization. The proof considers an arbitrary packing of congruent balls of radius 1 in Euclidean space and the properties that it must have to be a counterexample to the Kepler conjecture. Recall that we identify a packing of congruent balls with the discrete set V of centers of the balls.

The first step of the proof reduces the problem from a packing V involving a countably infinite number of congruent balls to a packing involving at most finitely many balls in close proximity to a fixed central ball. This reduction is obtained by making a geometric partition of Euclidean space that is adapted to the given packing. The pieces of this partition are called Marchal cells (replacing the decomposition stars in the original proof) [33].

A cell is not in general a polyhedron. An element of V that lies on the boundary of a cell is called a *vertex* of the cell. Each line segment between vertexes of the

cell that lies entirely on the boundary of the cell is called an *edge* of the cell. All the cells that share a *critical edge* (that is, an edge satisfying a certain restrictive length condition) form a *cell cluster* [17, Definition 6.89].

A real number $\Gamma(\varepsilon, X)$ is attached to each cell X together with one of its critical edges ε. This real number is defined geometrically in terms of the volume of the cell X, the solid angles at its vertexes, the dihedral angles along its edges, and the lengths of edges [17, Definitions 6.79, 6.91].

A key inequality (the cell-cluster inequality, which is discussed further in Section 4.4) provides the link between the Kepler conjecture and a local optimization problem (the local annulus inequality) involving at most finitely many balls [17, 6.93]. The cell-cluster inequality asserts that for every critical edge ε with corresponding cell cluster C,

$$\sum_{X \in C} \Gamma(\varepsilon, X) \geqslant 0.$$

A related inequality, which is easier to prove, holds when a cell does not have any critical edges [17, 6.92]. The significance of these inequalities is that they transform a problem about volumes and densities into a problem (the local annulus inequality) about distances between sphere centers.

The relationship among the cell-cluster inequality, the local annulus inequality, and the Kepler conjecture is expressed formally as a key intermediate result:

```
|- the_nonlinear_inequalities /\
   (!V. cell_cluster_inequality V) /\
   (!V. packing V /\ V SUBSET ball_annulus
        ==> local_annulus_inequality V)
   ==> the_kepler_conjecture
```

All three assumptions can be viewed as explicit optimization problems of continuous functions on compact spaces. The constant *ball annulus* is defined as the set $A = \{\mathbf{x} \in \mathbb{R}^3 : 2 \leqslant \|\mathbf{x}\| \leqslant 2.52\}$. As this set is compact and any packing V is discrete, the intersection of a packing with this set is necessarily finite. The local annulus inequality for a finite packing $V \subset A$ can be stated in the following form:

$$\sum_{\mathbf{v} \in V} f(\|\mathbf{v}\|) \leqslant 12, \tag{1}$$

where $f(t) = (2.52 - t)/(2.52 - 2)$ is the linear function that decays from 1 to 0 on the given annulus. An argument which we do not repeat here shows that it is enough to consider V with at most 15 elements [17, Lemma 6.110]. We discuss the proofs of the nonlinear inequalities and the cell-cluster inequality elsewhere in this paper. The local annulus inequality has been used to solve other open

problems in discrete geometry: Bezdek's strong dodecahedral conjecture and Fejes Tóth's contact conjecture [16].

The rest of the proof consists in proving the local annulus inequality. To carry this out, we assume that we have a counterexample V. The compactness of the ball annulus allows us to assume that the counterexample has a special form that we call *contravening*. We make a detailed study of the properties of a contravening V. The most important of these properties is expressed by what is called the main estimate (see Section 4.4). These properties imply that V gives rise to a combinatorial structure called a *tame planar hypermap*. (In this brief summary, we consider tame planar hypermaps to be essentially the same as plane graphs with certain restrictive properties. The nodes of the graph are in bijection with V.) A computer is used to enumerate all of the finitely many tame plane graphs up to isomorphism. This reduces the possible counterexamples V to an explicit finite family of cases.

For each explicitly given tame planar hypermap H, we may consider all contravening packings V associated with H. By definition, these are counterexamples to Equation (1). We express the conditions on V as a system of nonlinear inequalities. We relax the nonlinear system to a linear system and use linear programming techniques to show that the linear programs are infeasible and hence that the nonlinear system is inconsistent, so that the potential counterexample V cannot exist. Eliminating all possible counterexamples in this way, we conclude that the Kepler conjecture must hold.

4.3. Differences between the original proof and the blueprint proof. The blueprint proof follows the same general outline as the original proof. However, many changes have been made to make it more suitable for formalization. We list some of the primary differences between the two proofs

(1) In the blueprint proof, topological results concerning plane graphs are replaced with purely combinatorial results about hypermaps.

(2) The blueprint proof is based on a different geometric partition of space than that originally used. Marchal introduced this partition and first observed its relevance to the Kepler conjecture [33]. Marchal's partition is described by rules that are better adapted to formalization than the original.

(3) In a formal proof, every new concept comes at a cost: libraries of lemmas must be developed to support each concept. We have organized the blueprint proof around a small number of major concepts such as spherical trigonometry, volume, hypermap, fan, polyhedra, Voronoi partitions, linear programming, and nonlinear inequalities of trigonometric functions.

(4) The statements of the blueprint proof are more precise and make all hypotheses explicit.

(5) To permit a large collaboration, the chapters of the blueprint have been made as independent from one another as possible, and long proofs have been broken up into series of shorter lemmas.

(6) In the original, computer calculations were a last resort after as much was done by hand as feasible. In the blueprint, the use of computer has been fully embraced. As a result, many laborious lemmas of the original proof can be automated or eliminated altogether.

Because the original proof was not used for the formalization, we cannot assert that the original proof has been formally verified to be error free. Similarly, we cannot assert that the computer code for the original proof is free of bugs. The detection and correction of small errors is a routine part of any formalization project. Overall, hundreds of small errors in the proof of the Kepler conjecture were corrected during formalization.

4.4. Appendix to the blueprint. As part of the Flyspeck project, the blueprint proof has been supplemented with an 84 page unpublished appendix filled with additional details about the proof. We briefly describe the appendix.

The first part of the appendix gives details about how to formalize the main estimate [**17**, Section 7.4]. The main estimate is the most technically challenging part of the original text, and its formalization was the most technically challenging part of the blueprint text. In fact, the original proof of the main estimate contained an error that is described and corrected in [**19**]. This is the most significant error that was found.

A second major part of the appendix is devoted to the proof of the cell-cluster inequality [**17**, Theorem 6.93]. In the blueprint text, the proof is skipped, with the following comment: 'The proof of this cell-cluster inequality is a computer calculation, which is the most delicate computer estimate in the book. It reduces the cell-cluster inequality to hundreds of nonlinear inequalities in at most six variables.' The appendix gives full details.

A final part of the appendix deals with the final integration of the various libraries in the project. As a large collaborative effort that used two different proof assistants, there were many small differences between libraries that had to be reconciled to obtain a clean final theorem. The most significant difference to reconcile was the notion of planarity as used in classification of tame plane graphs and the notion of planarity that appears in the hypermap library. Until now, in our discussion, we have treated tame plane graphs and tame planar hypermaps as if there were essentially the same. However, in fact, they are based

on two very different notions of planarity. The recursive procedure defining tame graph planarity is discussed in Section 7.1. By contrast, in the hypermap library, the Euler characteristic formula is used as the basis of the definition of hypermap planarity. The appendix describes how to relate the two notions. This part of the appendix can be viewed as an expanded version of [**17**, Section 4.7.4].

4.5. Recording and replaying the proof. We also have an alternative weaker form of the main statement of the Kepler conjecture (from Section 3) that makes explicit all three computational portions of the proof. This form of the theorem leaves the list of graphs in the archive as an unspecified bound variable `a`. This weaker form has the advantage that it can be verified relatively quickly.

```
|- !a. tame_classification a /\
     good_linear_programming_results a /\
     the_nonlinear_inequalities
     ==> the_kepler_conjecture
```

We have employed an adapted version of HOL Light to record and export the formal proof steps generated by the proof scripts of the main statement in this form (www.proof-technologies.com/flyspeck/). The exported proof objects have been imported and executed in a separate HOL Light session involving just the HOL Light core system and a simple proof importing mechanism. They have also been imported and replayed in HOL Zero, another member of the HOL family [**1**].

This exercise has various benefits. First, it is generally much faster to import and replay a recorded formal proof than to execute the corresponding high-level proof script, whose execution typically involves a substantial amount of proof search beyond the actual formal proof steps. Second, it gives a further check of the formal proof. The successful import into HOL Zero, a system that pays particular attention to trustworthiness, effectively eliminates any risk of error in the proof. Finally, the exported proof objects are available to be imported and replayed in other HOL systems, opening the Flyspeck libraries of theorems to the users of other proof assistants.

5. Nonlinear inequalities

All but a few nonlinear inequalities in the Flyspeck project have the following general form

$$\forall \mathbf{x},\ \mathbf{x} \in D \implies f_1(\mathbf{x}) < 0 \vee \cdots \vee f_k(\mathbf{x}) < 0. \tag{2}$$

where $D = [a_1, b_1] \times \cdots \times [a_n, b_n]$ is a rectangular domain inside $\mathbb{R}^n$ and $\mathbf{x} = (x_1, \ldots, x_n)$. In the remaining few inequalities, the form is similar, except

that $k = 1$ and the inequality is not strict. In every case, the number of variables n is at most six. The following functions and operations appear in inequalities: basic arithmetic operations, square root, sine, cosine, arctangent, arcsine, arccosine, and the analytic continuation of $\arctan(\sqrt{x})/\sqrt{x}$ to the region $x > -1$. For every point $\mathbf{x} \in D$, at least one of the functions f_i is analytic in a neighborhood of $\mathbf{x}$ (and takes a negative value), but situations arise in which not every function is analytic or even defined at every point in the domain.

The formal verification of inequalities is based on interval arithmetic [35]. For example, $[3.14, 3.15]$ is an interval approximation of π since $3.14 \leqslant \pi \leqslant 3.15$. In order to work with interval approximations, arithmetic operations are defined over intervals. Denote the set of all intervals over $\mathbb{R}$ as $\mathbb{IR}$. A function (operation) $F : \mathbb{IR} \to \mathbb{IR}$ is called an interval extension of $f : \mathbb{R} \to \mathbb{R}$ if the following condition holds

$$\forall I \in \mathbb{IR}, \quad \{f(x) : x \in I\} \subset F(I).$$

This definition can be easily extended to functions on $\mathbb{R}^k$. It is easy to construct interval extensions of basic arithmetic operations and elementary functions. For instance,

$$[a_1, b_1] \oplus [a_2, b_2] = [a, b] \quad \text{for some } a \leqslant a_1 + a_2 \text{ and } b \geqslant b_1 + b_2.$$

Here, $\oplus$ denotes an interval extension of $+$. We do not define the result of $\oplus$ as $[a_1 + a_2, b_1 + b_2]$ since we may want to represent all intervals with limited precision numbers (for example, decimal numbers with at most 3 significant figures). With basic interval operations, it is possible to construct an interval extension of an arbitrary arithmetic expression by replacing all elementary operations with corresponding interval extensions. Such an interval extension is called the natural interval extension. Natural interval extensions can be imprecise, and there are several ways to improve them.

One simple way to improve upon the natural interval extension of a function is to subdivide the original interval into subintervals and evaluate an interval extension of the function on all subintervals. Using basic interval arithmetic and subdivisions, it would theoretically be possible to prove all nonlinear inequalities that arise in the project. This method does not work well in practice because the number of subdivisions required to establish some inequalities would be enormous, especially for multivariate inequalities.

Both the C++ informal verification code (from the original proof of the Kepler conjecture) and our formal verification procedure implemented in OCaml and HOL Light use improved interval extensions based on Taylor approximations.

Suppose that a function $g : \mathbb{R} \to \mathbb{R}$ is twice differentiable. Fix $y \in [a, b]$. Then we have the following formula for all $x \in [a, b]$:

$$g(x) = g(y) + g'(y)(x - y) + \tfrac{1}{2} g''(\xi)(x - y)^2$$

for some value $\xi = \xi(x) \in [a, b]$. Choose w such that $w \geqslant \max\{y - a, b - y\}$ and define $r(x) = |g'(y)|w + \frac{1}{2}|g''(\xi(x))|w^2$. We get the following inequalities

$$\forall x \in [a, b]. \quad g(y) - r(x) \leqslant g(x) \leqslant g(y) + r(x).$$

Let G, G_1, and G_2 be any interval extensions of g, g', and g''. Choose e such that $e \geqslant \operatorname{iabs}(G_1([y, y]))w + (w^2/2)\operatorname{iabs}(G_2([a, b]))$, where $\operatorname{iabs}([c, d]) = \max\{|c|, |d|\}$. Assume that $G([y, y]) = [g_l, g_u]$. Then the following function defines a (second order) Taylor interval approximation of $g(x)$:

$$G_{\text{Taylor}}([a, b]) = [l, u] \quad \text{where } l \leqslant g_l - e \text{ and } u \geqslant g_u + e.$$

That is,

$$\forall x \in [a, b]. \quad g(x) \in G_{\text{Taylor}}([a, b]).$$

In our verification procedure, we always take y close to the midpoint $(a + b)/2$ of the interval in order to minimize the value of w. There is an analogous Taylor approximation for multivariate functions based on the multivariate Taylor theorem.

As a small example, we compute a Taylor interval approximation of $g(x) = x - \arctan(x)$ on $[1, 2]$. We have $g'(x) = 1 - 1/(1 + x^2)$ and $g''(x) = -2x/(1 + x^2)^2$. Take $y = 1.5$ and natural interval extensions G_1 and G_2 of $g'(x)$ and $g''(x)$. Then $w = 0.5$, $G([1.5, 1.5]) = [0.517, 0.518]$, $G_1([1.5, 1.5]) = [0.692, 0.693]$, and $G_2([1, 2]) = [-0.5, -0.16]$. We get $e = 0.409$ and hence $G_{\text{Taylor}}([1, 2]) = [0.108, 0.927]$. This result is much better than the result obtained with the natural extension $G([1, 2]) = [-0.11, 1.22]$. We note that in the calculation of $G_{\text{Taylor}}([1, 2])$, we evaluate the expensive interval extension of arctan only once. To obtain similar accuracy with subdivision, more than one evaluation of arctan is needed. In general, it is necessary to subdivide the original interval into subintervals even when Taylor interval approximations are used. But in most cases, Taylor interval approximations lead to fewer subdivisions than natural interval extensions.

Taylor interval approximations may also be used to prove the monotonicity of functions. By expanding $g'(x)$ in a Taylor series, we obtain a Taylor interval approximation in a similar way:

$$\forall x \in [a, b]. \quad g'(x) \in G'_{\text{Taylor}}([a, b])$$

for some interval $G'_{\text{Taylor}}([a, b])$. If 0 is not in this interval, then the derivative has fixed sign, and the function g is monotonic, so that the maximum value of g occurs at the appropriate endpoint. More generally, in multivariate inequalities, a partial derivative of fixed sign may be used to reduce the verification on a rectangle of dimension k to an abutting rectangle of dimension $k - 1$.

A few of the inequalities are sharp. That is, the inequalities to be proved have the form $f \leqslant 0$, where $f(\mathbf{x}_0) = 0$ at some point $\mathbf{x}_0$ in the domain D of the inequality. In each case that arises, $\mathbf{x}_0$ lies at a corner of the rectangular domain. We are able to prove these inequalities by showing that (1) $f(\mathbf{x}_0) = 0$ by making a direct computation using exact arithmetic; (2) $f < 0$ on the complement of some small neighborhood U of $\mathbf{x}_0$ in the domain; and (3) every partial derivative of f on U has the appropriate sign to make the maximum of f on U to occur at $\mathbf{x}_0$. The final two steps are carried out using our standard tools of Taylor intervals.

All the ideas presented in the discussion above have been formalized in HOL Light and a special automatic verification procedure has been written in the combination of OCaml and HOL Light for verification of general multivariate nonlinear inequalities. This procedure consists of two parts. The first part is an informal search procedure which finds an appropriate subdivision of the original inequality domain and other information which can help in the formal verification step (such as whether or not to apply the monotonicity argument, which function from a disjunction should be verified, and so on). The second part is a formal verification procedure which takes the result of the informal search procedure as input and produces a final HOL Light theorem. A detailed description of the search and verification procedures can be found in [43, 45].

All formal numerical computations are done with special finite precision floating-point numbers formalized in HOL Light. It is possible to change the precision of all computations dynamically, and the informal search procedure tries to find minimal precision necessary for the formal verification. At the lowest level, all computations are done with natural numbers in HOL Light. We improved basic HOL Light procedures for natural numbers by representing natural numerals over an arbitrary base (the base 2 is the standard base for HOL Light natural numerals) and by providing arithmetic procedures for computing with such numerals. Note that all computations required in the nonlinear inequality verification procedure are done entirely inside HOL Light, and the results of all arithmetic operations are HOL Light theorems. The native floating-point operations of the computer are not used in any formal proof. As a consequence, the formal verification of nonlinear inequalities in HOL Light is much slower than the original informal C++ code.

The term `the_nonlinear_inequalities` is defined as a conjunction of several hundred nonlinear inequalities. The domains of these inequalities have been partitioned to create more than 23 000 inequalities. The verification of all nonlinear inequalities in HOL Light on the Microsoft Azure cloud took approximately 5000 processor hours. Almost all verifications were made in parallel with 32 cores using GNU parallel [46]. Hence the real time was less than a week ($5000 < 32 \times 168$ hours per week). These verifications were made

in July and August, 2014. Nonlinear inequalities were verified with compiled versions of HOL Light and the verification tool developed in Solovyev's 2012 thesis.

The verifications were rechecked at Radboud University on 60 hyperthreading Xeon 2.3GHz CPUs, in October 2014. This second verification required about 9370 processor hours over a period of six days. Identical results were obtained in these repeated calculations.

6. Combining HOL Light sessions

The nonlinear inequalities were obtained in a number of separate sessions of HOL Light that were run in parallel. By the design of HOL Light, it is not possible to pass a theorem from one session to another without fully reconstructing the proof in each session. To combine the results into a single session of HOL Light, we used a specially modified version of HOL Light that accepts a theorem from another session without proof. We briefly describe this modified version of HOL Light.

Each theorem is expressed by means of a collection of constants, and those constants are defined by other constants, recursively extending back to the primitive constants in the HOL Light kernel. Similarly, the theorem and constants have types, and those types also extend recursively back through other constants and types to the primitive types and constants of the kernel. A theorem relies on a list of axioms, which also have histories of constants and types. The semantics of a theorem is determined by this entire history of constants, types, and axioms, reaching back to the kernel.

The modified version of HOL Light is designed in such a way that a theorem can be imported from another session, provided the theorem is proved in another session, and the entire histories of constants, types, and axioms for that theorem are exactly the same in the two sessions. To implement this in code, each theorem is transformed into canonical form. To export a theorem, the canonical form of the theorem and its entire history are converted faithfully to a large string, and the MD5 hash of the string is saved to disk. The modified version of HOL Light then allows the import of a theorem if the appropriate MD5 is found.

To check that no pieces were overlooked in the distribution of inequalities to various cores, the pieces have been reassembled in the specially modified version of HOL Light. In that version, we obtain a formal proof of the theorem

```
|- the_nonlinear_inequalities
```

This theorem is exactly the assumption made in the formal proof of the Kepler conjecture, as stated in Section 3. We remark that the modified version of HOL

Light is not used during the proof of any other result of the Kepler conjecture. It is only used to assemble the small theorems from parallel sessions to produce this one master theorem.

7. Tame classification

The first major success of the Flyspeck project was the formalization of the classification of tame plane graphs. In the original proof of the Kepler conjecture, this classification was done by computer, using custom software to generate plane graphs satisfying given properties. This formalization project thus involved the verification of computer code. The formal verification of the code became the subject of Gertrud Bauer's PhD thesis under the direction of Nipkow. The work was completed by Nipkow [38].

As explained in Section 4, the tame plane graphs encode the possible counterexamples to the Kepler conjecture as plane graphs. The *archive* is a computer-generated list of all tame graphs. It is a text file that can be imported by the different parts of the proof. In this section we explain how the following completeness theorem is formalized in Isabelle/HOL:

|– "g ∈ PlaneGraphs" **and** "tame g" **shows** "fgraph g $\in_{\simeq}$ Archive"

The meaning of the terms `PlaneGraphs`, `tame`, `fgraph`, and `Archive` is explained in the following paragraphs. In informal terms, the completeness theorem asserts that every tame plane graph is isomorphic to a graph appearing in the archive.

Plane graphs are represented as an n-tuple of data including a list of faces. Faces are represented as lists of nodes, and each node is represented by an integer index. A function `fgraph` strips the n-tuple down to the list of faces.

To prove the completeness of the archive, we need to enumerate all tame plane graphs. For this purpose we rely on the fact that HOL contains a functional programming language. In essence, programs in HOL are simply sets of recursion equations, that is, pure logic. The original proof classifies tame plane graphs by a computer program written in Java. Therefore, as a first step in the formalization, we recast the original Java program for the enumeration of tame plane graphs in Isabelle/HOL. The result is a set of graphs called `TameEnum`. In principle we could generate `TameEnum` by formal proof but this would be extremely time and space consuming because of the huge number of graphs involved (see below). Therefore we rely on the ability of Isabelle/HOL to execute closed HOL formulas by translating them automatically into a functional programming language (in this case ML), running the program, and accepting the original formula as a theorem if the execution succeeds [12]. The programming language is merely used as a fast term rewriting engine.

We prove the completeness of the archive in two steps. First we prove that every tame plane graph is in `TameEnum`. This is a long interactive proof in Isabelle/HOL. Then we prove that every graph in `TameEnum` is isomorphic to some graph in the archive. Formally this can be expressed as follows:

```
|- fgraph ` TameEnum ⊆≃ Archive
```

This is a closed formula that Isabelle proves automatically by evaluating it (in ML) because all functions involved are executable.

In the following two subsections we give a high-level overview of the formalization and proof. For more details see [**36**, **38**]. The complete machine-checked proof, including the archive is available online in the Archive of Formal Proofs afp.sf.net [**5**]. Section 7.3 discusses the size of the archive and some performance issues.

7.1. Plane graphs and their enumeration. Plane graphs are not defined by the conventional mathematical definition. They are defined by an algorithm that starts with a polygon and inductively adds loops to it in a way that intuitively preserves planarity. (The algorithm is an implementation in code of a process of drawing a sequence of loops on a sheet of paper, each connected to the previous without crossings.)

Expressed as a computer algorithm, the enumeration of plane graphs proceeds inductively. It starts with a seed graph (the initial polygon) with two faces (intuitively corresponding to the two components in the plane of the complement of a Jordan curve), a final outer one and a nonfinal inner one, where a *final* face means that the algorithm is not permitted to make further modifications of the face. If a graph contains a nonfinal face, it can be subdivided into a final face and any number of nonfinal ones. Because a face can be subdivided in many ways, this process defines a forest of graphs. The leaves are final graphs. The formalization defines an executable function `next_plane` that maps a graph to the list of successor graphs reachable by subdividing one nonfinal face. The set of plane graphs, denoted `PlaneGraphs` in Isabelle/HOL, is defined to be the set of final graphs reachable from some seed graph in finitely many steps.

7.2. Tame graphs and their enumeration. The definition of tameness is already relatively close to an executable formulation. The two crucial constraints are that the faces of a tame graph may only be triangles up to hexagons, and that the 'admissible weight' of a tame graph is bounded from above. The tameness constraints imply that there are only finitely many tame plane graphs up to isomorphism (although we never need to prove this directly). The Isabelle/HOL predicate is called `tame`.

The enumeration of tame plane graphs is a modified enumeration of plane graphs where we remove final graphs that are definitely not tame, and prune nonfinal graphs that cannot lead to any tame graphs. The description in [14] is deliberately sketchy, but the Java programs provide precise pruning criteria. In the formalization we need to balance effectiveness of pruning with simplicity of the completeness proof: weak pruning criteria are easy to justify but lead to unacceptable run times of the enumeration, sophisticated pruning criteria are difficult to justify formally. In the end, for the formalization, we simplified the pruning criteria found in the Java code.

The formalization defines a function `next_tame` from a graph to a list of graphs. It is a restricted version of `next_plane` where certain kinds of graphs are removed and pruned, as described above. For computational reasons, the tameness check here is approximate: no tame graphs are removed, but nontame graphs may be produced. This is unproblematic: in the worst case a fake counterexample to the Kepler conjecture is produced, which is eliminated at a later stage of the proof, but we do not miss any real ones.

Each step of `next_tame` is executable. The exhaustive enumeration of all final graphs reachable from any seed graph via `next_tame` yields a set, denoted `TameEnum`.

7.3. The archive. The archive that came with the original proof contained (for historical reasons) over 5000 graphs. The first formalization [38] resulted in a reduced archive of 2771 graphs. During the completeness proof, the verified enumeration of tame graphs has to go through 2×10^7 intermediate and final graphs, which takes a few hours. With the advent of the blueprint proof, the definition of tameness changed. This change lead to 2×10^9 intermediate and final graphs and an archive with 18 762 graphs. The enumeration process had to be optimized to prevent it running out space and time [36]. In the end, the enumeration again runs in a few hours. The formalization uncovered and fixed a bug in the original Java program, related to symmetry optimizations. Fortunately, this bug was not executed in the original proof, so that the output was correct. However, the bug caused two graphs to be missed in an early draft of the blueprint proof.

8. Importing results from Isabelle

The tame graph classification was done in the Isabelle/HOL proof assistant, while all the rest of the project has been carried out in HOL Light. It seems that it would be feasible to translate the Isabelle code to HOL Light to have the entire project under the same roof, but this lies beyond the scope of the Flyspeck project.

Current tools do not readily allow the automatic import of this result from Isabelle to HOL Light. A tool that automates the import from Isabelle to HOL Light was written by McLaughlin with precisely this application in mind [34], but this tool has not been maintained. A more serious issue is that the proof in Isabelle uses computational reflection as described at the end of Section 2, but the HOL Light kernel does not permit reflection. Thus, the reflected portions of the formal proof would have to be modified as part of the import.

Instead, we leave the formalization of the Kepler conjecture distributed between two different proof assistants. In HOL Light, the Isabelle work appears as an assumption, expressed through the following definition.

```
|- import_tame_classification <=>
     (!g. g IN PlaneGraphs /\ tame g ==>
      fgraph g IN_simeq archive)
```

The left-hand side is exactly the assumption made in the formal proof of the Kepler conjecture, as stated in Section 3. The right-hand side is the verbatim translation into HOL Light of the following completeness theorem in Isabelle (repeated from above):

|– ”g ∈ PlaneGraphs” **and** ”tame g” **shows** ”fgraph g $\in_{\simeq}$ Archive”

All of the HOL Light terms `PlaneGraphs`, `tame`, `archive`, `IN_simeq`, `fgraph` are verbatim translations of the corresponding definitions in Isabelle (extended recursively to the constants appearing in the definitions). The types are similarly translated between proof assistants (lists to lists, natural numbers to natural numbers, and so forth). These definitions and types were translated by hand. The archive of graphs is generated from the same ML file for both the HOL Light and the Isabelle statements.

There are some insignificant differences between the Isabelle definitions and the HOL Light translation. Some specifications leave some undefined cases, and we do not attempt a verbatim translation of undefined behavior. For example, the head of an empty list is unspecified, and the two proof assistants might give inequivalent behavior for the head of an empty list. Such differences are irrelevant to the formal proof, which operates at an abstract level that avoids reasoning about undefined behavior.

There are some insignificant differences between the types. For example, a data type with two constructors in Isabelle is implemented in HOL Light as the boolean type:

datatype facetype = Final | Nonfinal

Since the formal proof is distributed between two different systems with two different logics, we indicate why this theorem in Isabelle must also be a theorem

in HOL Light. Briefly, this particular statement could be expressed as a SAT problem in first-order propositional logic. SAT problems pass directly between systems and are satisfiable in one system if and only if they are satisfiable in the other (assuming the consistency of both systems). In expressing the classification theorem as a SAT problem, the point is that all quantifiers in the theorem run over bounded discrete sets, allowing them to be expanded as finitely many cases in propositional logic.

In more detail, each graph that occurs in the tame classification has at most fifteen nodes. Each of the finitely many possible graphs can be specified by a finite number of boolean conditions. Similarly, graph isomorphism can be replaced by an enumeration of finitely many possible bijections of nodes. Tameness involves some conditions in integer arithmetic, but again everything is bounded, and a bounded arithmetic table can be constructed in boolean gates using the usual tricks. Thus, at a theoretical level, a SAT problem is solved, which can be shared between systems.

More generally, the logics of Isabelle/HOL and HOL Light are very closely related to each another, and they implement the same computation rules. If computable predicates are implemented in the two proof assistants using the same functions and data types, then the results of the computations must agree. In this way, computable predicates can pass from Isabelle to HOL Light.

9. Linear programs

We return to the sketch of the proof from Section 4 and add some details about the role of linear programming. The blueprint proof reduces infinite packings that are potential counterexamples to the Kepler conjecture to finite packings V, called contravening packings. The combinatorial structure of a contravening packing can be encoded as a tame planar hypermap. The tame graph classification theorem implies that there are finitely many tame planar hypermaps up to isomorphism. For each such hypermap it is possible to generate a list of inequalities which must be satisfied by a potential counterexample associated with the given tame planar hypermap. Most of these inequalities are nonlinear.

To rule out a potential counterexample, it is enough to show that the system of nonlinear inequalities has no solution. A linear relaxation of these inequalities is obtained by replacing nonlinear quantities with new variables. For instance, the dihedral angles of a simplex are nonlinear functions of the edge lengths of the simplex; new variables are introduced for each dihedral angle to linearize any expression that is linear in the angles. The next step is to show that the linear program is not feasible. Infeasibility implies that the original system of

nonlinear inequalities is inconsistent, and hence there is no contravening packing associated with the given tame planar hypermap. The process of construction and solution of linear programs is repeated for all tame planar hypermaps and it is shown that no contravening packings (and hence no counterexamples to the Kepler conjecture) exist.

There are two parts in the formal verification of linear programs. First, linear programs are generated from formal theorems, nonlinear inequalities, and tame planar hypermaps. Second, a general linear program verification procedure verifies all generated linear programs.

The formal generation of linear programs follows the procedure outlined above. The first step is generation of linear inequalities from properties of contravening packings. Many such properties directly follow from nonlinear inequalities. As described above, nonlinear inequalities are transformed into linear inequalities by introducing new variables for nonlinear expressions. In fact, we do not change the original nonlinear inequalities but simply introduce new HOL Light constants for nonlinear functions and reformulate nonlinear inequalities in a linear form.

For example, suppose we have the following inequalities: $x+x^2 \leqslant 3$ and $x \geqslant 2$. Define $y = x^2$ to obtain the following linear system of inequalities: $x + y \leqslant 3$, $x \geqslant 2$, and $y \geqslant 4$. (The last inequality follows from $x \geqslant 2$.) We ignore the nonlinear dependency of y on x, and obtain a system of linear inequalities which can be easily shown to be inconsistent.

The generation of linear inequalities from properties of contravening packings is a semiautomatic procedure: many such inequalities are derived with a special automatic procedure but some of them need manual formal proofs.

The next step is the relaxation of linear inequalities with irrational coefficients. We do not solve linear programs with irrational numbers directly. Consider the example, $x - \sqrt{2}y \leqslant \pi$, $x + y \leqslant \sqrt{35}$, $x \geqslant 5$, $y \geqslant 0$. These inequalities imply

$$x \geqslant 5, \quad y \geqslant 0, \quad x - 1.42y \leqslant 3.15, \quad x + y \leqslant 6.$$

This system with rational coefficients is inconsistent and thus the original system is inconsistent.

The relaxation of irrational coefficients is done completely automatically. In fact, inequalities with integer coefficients are produced by multiplying each inequality by a sufficiently large power of 10 (decimal numerals are used in the verification of linear programs).

The last step of the formal generation of linear programs is the instantiation of free variables of linear inequalities with special values computed from an associated tame planar hypermap. In this way, each tame planar hypermap produces a linear program which is formally checked for feasibility. Not all

linear programs obtained in this way are infeasible. For about half of the tame planar hypermaps, linear relaxations are insufficiently precise and do not yield infeasible linear programs. In that situation, a feasible linear program is split into several cases where new linear inequalities are introduced. New cases are produced by considering alternatives of the form $x \leqslant a \vee a \leqslant x$ where x is some variable and a is a constant. Case splitting leads to more precise linear relaxations.

Case splitting for verification of linear programs in the project is automatic. In fact, a special informal procedure generates all required cases first, and the formal verification procedure for linear programs is applied to each case.

Formal verification of general linear programs is relatively easy. We demonstrate our verification procedure with an example. A detailed description of this procedure can be found in [44]. All variables in each verified linear program must be nonnegative and bounded. In the following example, our goal is to verify that the following system is infeasible:

$$x - 1.42y \leqslant 3.15, \quad x + y \leqslant 6, \quad 5 \leqslant x \leqslant 10, \quad 0 \leqslant y \leqslant 10.$$

Introduce slack variables s_1, s_2 for inequalities which do not define bounds of variables, and construct the following linear program:

$$\begin{aligned} &\text{minimize } s_1 + s_2 \text{ subject to} \\ &x - 1.42y \leqslant 3.15 + s_1, \quad x + y \leqslant 6 + s_2, \quad 5 \leqslant x \leqslant 10, \\ &0 \leqslant y \leqslant 10, \quad 0 \leqslant s_1,\ 0 \leqslant s_2. \end{aligned} \tag{3}$$

If the objective value of this linear program is positive then the original system is infeasible. We are interested in a dual solution of this linear program. Dual variables correspond to the constraints of the primal linear program. We use an external linear programming tool for finding dual solutions, which may be imprecise. A procedure, which is described in [44], starts with an initial imprecise dual solution and modifies it to obtain a carefully constructed dual solution that has sufficient numerical precision to prove the infeasibility of the original (primal) system. In the example at hand, we get the following modified dual solution:

$$(0.704, 1, -1.705, 0.001, -0.00032, 0, 0.296, 0),$$

whose entries correspond with the ordered list of eight inequalities in the system (3). With this modified dual solution, the sum of the constraints of the system (3) with the slack variables set to zero, weighted by the coefficients of the dual vector, yields $0x + 0y \leqslant -0.2974$. This contradiction shows that our original system of inequalities is infeasible. We stress that the coefficients of x and y

in this sum are precisely zero, and this is a key feature of the modified dual solutions.

If we know a modified dual solution, then the formal verification reduces to the summation of inequalities with coefficients from the modified dual solution and checking that the final result is inconsistent. A modified dual solution can be found with informal methods. We use GLPK for solving linear programs [8] and a special C# program for finding the required modified dual solutions for linear programs. All dual solutions are also converted to integer solutions by multiplying all coefficients by a sufficiently large power of 10. The formal verification procedure uses formal integer arithmetic. There are 43 078 linear programs (after considering all possible cases). All these linear programs can be verified in about 15 h on a 2.4 GHz computer. The verification time does not include time for generating modified dual solutions. These dual solutions need only be computed once and saved to files that are later loaded by the formal verification procedure.

10. Auditing a distributed formal proof

A proof assistant largely cuts the mathematical referees out of the verification process. This is not to say that human oversight is no longer needed. Rather, the nature of the oversight is such that specialized mathematical expertise is only needed for a small part of the process. The rest of the audit of a formal proof can be performed by any trained user of the HOL Light and Isabelle proof assistants.

Adams [2] describes the steps involved in the auditing of this formal proof. The proof scripts must be executed to see that they produce the claimed theorem as output. The definitions must be examined to see that the meaning of the final theorem (the Kepler conjecture) agrees with the common understanding of the theorem. In other words, did the right theorem get formalized? Were any unapproved axioms added to the system? The formal proof system itself should be audited to make sure there is no foul play in the syntax, visual display, and underlying internals. A fraudulent user of a proof assistant might ‘exploit a flaw to get the project completed on time or on budget. In their review, the auditor must assume malicious intent, rather than use arguments about the improbability of innocent error’ [2].

This particular formal proof has several special features that call for careful auditing. The most serious issue is that the full formal proof was not obtained in a single session of HOL Light. An audit should check that the statement of the tame classification theorem in Isabelle has been faithfully translated into HOL Light. (It seems to us that our single greatest vulnerability to error lies in the hand translation of this one statement from Isabelle to HOL Light, but even here there

is no mathematical reasoning involved beyond a rote translation.) In particular, an audit should check that the long list of tame graphs that is used in Isabelle is identical to the list that is used in HOL Light. (As mentioned above, both systems generated their list from the same master file.)

The auditor should also check the design of the special modification of HOL Light that was used to combine nonlinear inequalities into a single session.

11. Related work

Numerous other research projects in formal proofs have made use of the Flyspeck project in some way or have been inspired by the needs of Flyspeck. These projects include the work cited above on linear programming and formal proofs of nonlinear inequalities, automated translation of proofs between formal proof systems [24, 34, 42], the refactoring of formal proofs [3], machine learning applied to proofs and proof automation [25, 26], an SSReflect mode for HOL Light [43], and a mechanism to execute trustworthy external arithmetic from Isabelle/HOL [40, 41].

The roles of the various members of the Flyspeck project have been spelled out in the email announcement of the project completion, posted at [7].

Acknowledgements

We wish to acknowledge the help, support, influence, and various contributions of the following individuals: Nguyen Duc Thinh, Nguyen Duc Tam, Vu Quang Thanh, Vuong Anh Quyen, Catalin Anghel, Jeremy Avigad, Henk Barendregt, Herman Geuvers, Georges Gonthier, Daron Green, Mary Johnston, Christian Marchal, Laurel Martin, Robert Solovay, Erin Susick, Dan Synek, Nicholas Volker, Matthew Wampler-Doty, Benjamin Werner, Freek Wiedijk, Carl Witty, and Wenming Ye.

We wish to thank the following sources of institutional support: NSF grant 0503447 on the 'Formal Foundations of Discrete Geometry' and NSF grant 0804189 on the 'Formal Proof of the Kepler Conjecture,' Microsoft Azure Research, William Benter Foundation, University of Pittsburgh, Radboud Research Facilities, Institute of Math (VAST), and VIASM.

We thank anonymous referees for rerunning the proof scripts, for improvements to the project configuration, and other suggestions.

Supplementary material

The code and documentation for the Flyspeck project are available at: https://github.com/flyspeck/flyspeck.

Appendix on definitions

The following theorem provides evidence that key definitions in the statement of the Kepler conjecture are the expected ones.

```
|-
// real absolute value:
   (&0 <= x ==> abs x = x) /\
   (x < &0 ==> abs x = -- x) /\

// powers:
    x pow 0 = &1 /\ x pow SUC n = x * x pow n /\

// square root:
   (&0 <= x ==> &0 <= sqrt x /\ sqrt x pow 2 = x) /\

// finite sums:
   sum (0..0) f = f 0 /\ sum (0..SUC n) f =
     sum (0..n) f + f (SUC n) /\

// pi:
   abs (pi / &4 - sum (0..n)
     (\i. (-- &1) pow i / &(2 * i + 1)))
     <= &1 / &(2 * n + 3) /\

// finite sets and their cardinalities:
   (A HAS_SIZE n <=> FINITE A /\ CARD A = n) /\
   {} HAS_SIZE 0 /\ {a} HAS_SIZE 1 /\
   (A HAS_SIZE m /\ B HAS_SIZE n /\
   (A INTER B) HAS_SIZE p
     ==> (A UNION B) HAS_SIZE (m+n - p)) /\

// bijection between R^3 and ordered triples of reals:
   triple_of_real3 o r3 = (\w:real#real#real. w) /\
   r3 o triple_of_real3 = (\v:real^3. v) /\

// the origin:
   vec 0 = r3(&0,&0,&0) /\

// the metric on R^3:
   dist(r3(x,y,z),r3(x',y',z')) =
     sqrt((x - x') pow 2 + (y - y') pow 2 + (z - z') pow 2) /\

// a packing:
   (packing V <=>
     (!u v. u IN V /\ v IN V /\ dist(u,v) < &2 ==> u = v))
```

References

[1] M. Adams, 'Introducing HOL Zero', in *Mathematical Software–ICMS 2010* (Springer, 2010), 142–143.

[2] M. Adams, 'Flyspecking Flyspeck', in *Mathematical Software–ICMS 2014* (Springer, 2014), 16–20.

[3] M. Adams and D. Aspinall, 'Recording and refactoring HOL Light tactic proofs', in *Proceedings of the IJCAR Workshop on Automated Theory Exploration* (2012).

[4] G. Bauer, 'Formalizing plane graph theory – towards a formalized proof of the Kepler conjecture'. PhD Thesis, Technische Universität München, 2006. http://mediatum.ub.tum.de/doc/601794/document.pdf.

[5] G. Bauer and T. Nipkow, 'Flyspeck I: Tame graphs', in *The Archive of Formal Proofs* (eds. G. Klein, T. Nipkow and L. Paulson) http://afp.sf.net/entries/Flyspeck-Tame.shtml, May 2006. Formal proof development.

[6] L. Fejes Tóth, *Lagerungen in der Ebene auf der Kugel und im Raum*, 1st edn, (Springer, Berlin–New York, 1953).

[7] Flyspeck, The Flyspeck Project, 2014. https://github.com/flyspeck/flyspeck.

[8] GLPK, GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/.

[9] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor and S. O. Biha *et al.*, 'A machine-checked proof of the odd order theorem', in *Interactive Theorem Proving* (Springer, 2013), 163–179.

[10] M. Gordon, R. Milner and C. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, Lecture Notes in Computer Science, 78 (1979).

[11] M. J. C. Gordon and T. F. Melham, *Introduction to HOL: a Theorem Proving Environment for Higher Order Logic* (Cambridge University Press, 1993).

[12] F. Haftmann and T. Nipkow, 'Code generation via higher-order rewrite systems', in *Functional and Logic Programming, 10th International Symposium, FLOPS 2010,* Sendai, Japan, April 19–21, 2010. *Proceedings* (Springer, 2010), 103–117. https://dx.doi.org/10.1007/978-3-642-12251-4_9.

[13] T. Hales, Developments in formal proofs. *Bourbaki Seminar*, 2013/2014 (1086):1086-1-23, 2014.

[14] T. C. Hales, 'A proof of the Kepler conjecture', *Ann. of Math. (2)* **162** (2005), 1063–1183.

[15] T. C. Hales, 'Introduction to the Flyspeck Project', in *Mathematics, Algorithms, Proofs* (eds. T. Coquand, H. Lombardi and M.-F. Roy) Dagstuhl Seminar Proceedings, Dagstuhl, Germany, number 05021 (Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006), http://drops.dagstuhl.de/opus/volltexte/2006/432.

[16] T. C. Hales, 'The strong dodecahedral conjecture and Fejes Tóth's contact conjecture', Technical Report, 2011.

[17] T. C. Hales, *Dense Sphere Packings: a Blueprint for Formal Proofs*, London Mathematical Society Lecture Note Series, 400 (Cambridge University Press, 2012).

[18] T. C. Hales and S. P. Ferguson, 'The Kepler conjecture', *Discrete Comput. Geom.* **36**(1) (2006), 1–269.

[19] T. C. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua and R. Zumkeller, 'A revision of the proof of the Kepler Conjecture', *Discrete Comput. Geom.* **44**(1) (2010), 1–34.

[20] J. Harrison, 'Towards self-verification of HOL Light', in *Automated Reasoning, Third International Joint Conference, IJCAR 2006,* Seattle, WA, USA, August 17–20, 2006, *Proceedings* (eds. U. Furbach and N. Shankar) Lecture Notes in Computer Science, 4130 (Springer, 2006), 177–191. ISBN 3-540-37187-7. https://dx.doi.org/10.1007/11814771_17.

[21] J. Harrison, 'Without loss of generality', in *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009* (eds. S. Berghofer, T. Nipkow, C. Urban and M. Wenzel) Lecture Notes in Computer Science, 5674 (Springer, Munich, Germany, 2009), 43–59.
[22] J. Harrison, 'HOL Light: An overview', in *Theorem Proving in Higher Order Logics* (Springer, 2009), 60–66.
[23] J. Harrison, The HOL Light theorem prover, 2014. http://www.cl.cam.ac.uk/~jrh13/hol-light/index.html.
[24] C. Kaliszyk and A. Krauss, 'Scalable LCF-style proof translation', in *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)* (eds. S. Blazy, C. Paulin-Mohring and D. Pichardie) Lecture Notes in Computer Science, 7998 (Springer, 2013), 51–66.
[25] C. Kaliszyk and J. Urban, 'Learning-assisted automated reasoning with Flyspeck', *J. Automat. Reason.* **53**(2) (2014), 173–213. https://dx.doi.org/10.1007/s10817-014-9303-3.
[26] C. Kaliszyk and J. Urban, 'Learning-assisted theorem proving with millions of lemmas', *J. Symbolic Comput.* **69**(0) (2014), 109–128. ISSN 0747-7171. doi:10.1016/j.jsc.2014.09.032. URL http://www.sciencedirect.com/science/article/pii/S074771711400100X.
[27] J. Kepler, *Strena seu de nive sexangula* (Gottfried. Tampach, Frankfurt, 1611).
[28] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood, 'seL4: formal verification of an OS kernel', in *Proc. 22nd ACM Symposium on Operating Systems Principles 2009* (eds. J. N. Matthews and T. E. Anderson) (ACM, 2009), 207–220.
[29] R. Kumar, R. Arthan, M. O. Myreen and S. Owens, 'HOL with definitions: semantics, soundness, and a verified implementation', in *Interactive Theorem Proving – 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014,* Vienna, Austria, July 14–17, 2014. *Proceedings* (eds. G. Klein and R. Gamboa) Lecture Notes in Computer Science, 8558 (Springer, 2014), 308–324. ISBN 978-3-319-08969-0. https://dx.doi.org/10.1007/978-3-319-08970-6_20.
[30] J. Lagarias, *The Kepler Conjecture and its Proof* (Springer, 2009), 3–26.
[31] X. Leroy, 'Formal certification of a compiler back-end, or: programming a compiler with a proof assistant', in *ACM SIGPLAN Notices* 41, (2006), 42–54. http://compcert.inria.fr/.
[32] V. Magron, 'Formal proofs for global optimization – templates and sums of squares'. PhD Thesis, École Polytechnique, 2013.
[33] C. Marchal, 'Study of the Kepler's conjecture: the problem of the closest packing', *Math. Z.* **267**(3–4) (2011), 737–765. ISSN 0025-5874. https://dx.doi.org/10.1007/s00209-009-0644-2.
[34] S. McLaughlin, 'An interpretation of Isabelle/HOL in HOL Light', in *IJCAR* (eds. U. Furbach and N. Shankar) Lecture Notes in Computer Science, 4130 (Springer, 2006), 192–204.
[35] R. E. Moore, R. B. Kearfott and M. J. Cloud, *Introduction to Interval Analysis* (SIAM, 2009).
[36] T. Nipkow, 'Verified efficient enumeration of plane graphs modulo isomorphism', in *Interactive Theorem Proving (ITP 2011)* (eds. M. Van Eekelen, H. Geuvers, J. Schmaltz and F. Wiedijk) Lecture Notes in Computer Science, 6898 (Springer, 2011), 281–296.
[37] T. Nipkow, L. Paulson and M. Wenzel, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, Lecture Notes in Computer Science, 2283 (Springer, 2002), http://www.in.tum.de/~nipkow/LNCS2283/.

[38] T. Nipkow, G. Bauer and P. Schultz, 'Flyspeck I: Tame graphs', in *Automated Reasoning (IJCAR 2006)* (eds. U. Furbach and N. Shankar) Lecture Notes in Computer Science, 4130 (Springer, 2006), 21–35.
[39] S. Obua, 'Proving bounds for real linear programs in Isabelle/HOL', in *Theorem Proving in Higher Order Logics* (eds. J. Hurd and T. F. Melham) Lecture Notes in Computer Science, 3603 (Springer, 2005), 227–244.
[40] S. Obua, 'Flyspeck II: the basic linear programs'. PhD Thesis, Technische Universität München, 2008.
[41] S. Obua and T. Nipkow, 'Flyspeck II: the basic linear programs', *Ann. Math. Artif. Intell.* **56**(3–4) (2009).
[42] S. Obua and S. Skalberg, 'Importing HOL into Isabelle/HOL', in *Automated Reasoning*, Lecture Notes in Computer Science, 4130 (Springer, 2006), 298–302.
[43] A. Solovyev, 'Formal methods and computations'. PhD Thesis, University of Pittsburgh, 2012. http://d-scholarship.pitt.edu/16721/.
[44] A. Solovyev and T. C. Hales, *Efficient Formal Verification of Bounds of Linear Programs*, Lecture Notes in Computer Science, 6824 (Springer, 2011), 123–132.
[45] A. Solovyev and T. C. Hales, 'Formal verification of nonlinear inequalities with Taylor interval approximations', in *NFM*, Lecture Notes in Computer Science, 7871 (Springer, 2013), 383–397.
[46] O. Tange, 'GNU parallel – the command-line power tool', *USENIX Mag.* **36**(1) (2011), 42–47. URL http://www.gnu.org/s/parallel.
[47] C. Tankink, C. Kaliszyk, J. Urban and H. Geuvers, 'Formal mathematics on display: A wiki for Flyspeck', in *MKM/Calculemus/DML* (eds. J. Carette, D. Aspinall, C. Lange, P. Sojka and W. Windsteiger) Lecture Notes in Computer Science, 7961 (Springer, 2013), 152–167. ISBN 978-3-642-39319-8.
[48] R. Zumkeller, 'Global optimization in type theory'. PhD Thesis, École Polytechnique Paris, 2008.