hexagon:2611.00394v1

A FORMAL PROOF IN LEAN OF THE FINITENESS OF PLANAR FIVE-BODY CENTRAL CONFIGURATIONS FOR ALL POSITIVE MASSES

TEJASVI SINGH TOMAR

Abstract. Smale's sixth problem asks whether, for every choice of positive masses, there are finitely many central configurations up to similarity. For five bodies in the plane, Albouy and Kaloshin proved finiteness outside a set of masses of codimension two. In a companion paper the author proved finiteness for every choice of five positive masses by their method, adding steps that cannot be checked by hand: exact computer algebra, rational certificates and a branch and bound. We formalize this theorem in Lean 4 with Mathlib without assuming the results of Albouy and Kaloshin that the informal proof uses. Instead we prove what the proof needs of their rules on limit diagrams, of their classification of these diagrams and of their mass conditions. We do so at places of complex algebraic curves, taken as valuations of the function field rather than points of a projective model. Almost every computation becomes data for a checker that we write in Lean, prove sound with respect to Mathlib's real and complex numbers, and run in the kernel (proof by reflection). The main theorem depends on about 680,000 theorems proved in this way, and Mathlib's tactics prove the few other computations. The proof uses only Lean's three standard axioms, and two independent Lean kernels, nanoda and lean4lean, accept it.

1. Introduction

A configuration $q = (q_1, \dots, q_n)$ of distinct points in the plane, with positive masses $m_1, \dots, m_n$, is a *central configuration* if there are $\lambda \in \mathbb{R}$ and $c \in \mathbb{R}^2$ such that

$$\lambda\, m_i(q_i - c) = \sum_{j \neq i} \frac{m_i m_j\,(q_i - q_j)}{|q_i - q_j|^3} \qquad (i = 1, \dots, n). \tag{1}$$

Central configurations are the shapes of homographic solutions in the Newtonian n-body problem, and a configuration similar to a central configuration is again one. Smale's sixth problem [21] asks whether, for every choice of positive masses, there are finitely many central configurations up to similarity. Hampton and Moeckel [11] proved finiteness for four bodies. For five bodies in the plane, Albouy and Kaloshin [1] proved it when the masses lie outside a closed algebraic subset of codimension two. The author's paper [24] removes the exceptional set.

Theorem 1.1 ([24, Theorem 1.1])**.** *For every $m \in (\mathbb{R}_{>0})^5$ the planar five-body problem with masses m has finitely many central configurations up to similarity.*

Z. Chen and Z. Tong [6] state the same theorem and give a different proof, which, as far as we know, has not been formalized.

The method of [1] starts from the fact that infinitely many central configurations would give a curve of complex solutions, at whose places at infinity the valuations of the mutual distances must balance; this is the balancing condition of [1]. The proof in [24] follows this method, but it adds two kinds of steps that a reader cannot check by hand. First, there is exact computer algebra. Standard bases and minimal primes over $\mathbb{Q}$ give ideals on sixteen strata, which describe the sets of mass conditions that can hold together at positive masses [24, §8]. Then rational

2020 *Mathematics Subject Classification.* 68V20, 68V15, 70F10, 70F15, 14H05.

Key words and phrases. Central configurations, five-body problem, Smale's sixth problem, Lean, Mathlib, proof by reflection, computer-assisted proof, formal verification, valuations.

certificates show that the balancing condition of [1] fails under the resulting sets of conditions, except at two mass points and on a few one-parameter families of masses [24, Theorem 8.2]. Further certificates treat the two mass points, relative to a smaller hull of diagram 5, and the one-parameter families. At finitely many points of the families they use Gröbner bases modulo large primes with a lifting argument. Second, there is a counting step. On the rays through $(1, 4, 4, 4, 4)$ and $(1, 1, 4, 4, 4)$, and at one further mass point, the certificates do not apply. There the real central configurations are counted by a branch and bound in outward-rounded floating point [24, §10]. Independent programs re-check most of these computations [24, §12], but they still rely on computer algebra systems and on floating-point code.

We formalized the whole proof in Lean 4 [8] with Mathlib [17]. The theorem `CC5Full.Core.Compose.MainTheorem_holds` proves the formal statement `CC5Full.MainTheorem` of Theorem 1.1 with no hypotheses, using only the three standard axioms (§2.4). We do not assume the inputs from [1] that are listed in [24, §2.6]. Data produced by our programs enter only as certificates checked by the Lean kernel. Almost all of them go through checkers that we have proved sound in Lean, and a few polynomial identities and inequalities through Mathlib's tactics. The main theorem and the declarations on which it depends form its closure. Lean's kernel checks each of them when its module is built, and the independent kernels nanoda [20] and lean4lean [5] check the whole closure again from exports (§6).

We write for a reader who knows the mathematics. In §2 we state the formal theorem and compare it with Theorem 1.1. In §3 we describe how the informal proof maps onto the development, with a table that links the main results of [24] to Lean statements and their proofs, and we list where the formal proof departs from [24]. The new parts of the development are the subject of §§4–6: a theory of places of complex algebraic curves, built on Mathlib, in which we prove the inputs from [1] (§4); checkers for the certificates, written so that the kernel can evaluate them at this scale (§5); and the check of the whole closure by two independent kernels, which revealed an issue in one of them (§6). In §7 we describe how we reproduce the build, with all checks collected in Table 5, and in §8 we compare the project with other formal proofs that depend on large computations.

Citations and files. We cite the statements of [24] by their numbers in its version 2. File names refer to the Lean project `lean_full` (package `CC5Full`) at commit `52858e1`, in the supplementary material [25]. Paths in tables are relative to the project directory `CC5Full/` unless they start with it.

2. The formal statement

Lean checks the proof of the formal statement, but only a reader can check that the statement is the intended theorem. Translations between informal and formal mathematics can fail in subtle ways [3]. We therefore print the statement in full, compare it with Theorem 1.1 line by line, and say exactly what has to be trusted.

2.1. **The definitions.** In Listing 1 we copy the definitions of `CC5Full/System.lean`, lines 38–66, with `Masses` from `CC5Full/Basic.lean`, line 26, and the final theorem. `MainTheorem` uses twelve lines of these definitions. We also show `SimilarOP`, the orientation-preserving relation, which `MainTheorem` does not use. After `import CC5Full.Core.Compose.Main`, the command `#print CC5Full.IsCC` and its analogues print the elaborated definitions, in which the casts appear explicitly. With the option `pp.all`, Lean prints them without any notation.

2.2. **Line by line.** In Table 1 we match each part of Theorem 1.1, and of equation (1), which defines central configurations, with its counterpart in Listing 1.

LISTING 1. The formal statement of Theorem 1.1 (`Basic.lean`, line 26; `System.lean`, lines 38–66) and its proof (`Core/Compose/Main.lean`, line 48), without the docstrings. Here `univ.erase i` is the set of indices $j \neq i$. The map `starRingEnd` $\mathbb{C}$ is complex conjugation. `MainTheorem` does not use `SimilarOP`.

```
abbrev Masses := Fin 5 → ℝ
abbrev Conf := Fin 5 → ℂ
def CollisionFree (q : Conf) : Prop := ∀ i j, i ≠ j → q i ≠ q j
def IsCC (m : Masses) (q : Conf) : Prop :=
  CollisionFree q ∧ ∃ (lam : ℝ) (c : ℂ), ∀ i,
    ((lam * m i : ℝ) : ℂ) * (q i - c) =
      ∑ j ∈ univ.erase i, ((m i * m j / ‖q i - q j‖ ^ 3 : ℝ) : ℂ) * (q i - q j)
def Similar (q q' : Conf) : Prop :=
  ∃ a b : ℂ, a ≠ 0 ∧ ((∀ i, q' i = a * q i + b) ∨ (∀ i, q' i = a * starRingEnd ℂ (q i) + b))
def SimilarOP (q q' : Conf) : Prop := ∃ a b : ℂ, a ≠ 0 ∧ ∀ i, q' i = a * q i + b
def FinitelyManyCC (m : Masses) : Prop :=
  ∃ F : Finset Conf, ∀ q, IsCC m q → ∃ q' ∈ F, Similar q' q
def MainTheorem : Prop := ∀ m : Masses, (∀ i, 0 < m i) → FinitelyManyCC m

theorem MainTheorem_holds : MainTheorem := MainTheorem_of_interface_holds Interface_holds
```

TABLE 1. Theorem 1.1 and equation (1) against the formal definitions.

Theorem 1.1 and (1)	Listing 1
five bodies $1, \dots, 5$	the elements `0`, …, `4` of `Fin 5`
masses $m \in (\mathbb{R}_{>0})^5$	`m : Masses`, a function `Fin 5` $\to \mathbb{R}$, with the hypothesis `∀ i, 0 < m i` of `MainTheorem`
a configuration q in the plane $\mathbb{R}^2$	`q : Conf`, a function `Fin 5` $\to \mathbb{C}$
of distinct points	`CollisionFree q`
there are $\lambda \in \mathbb{R}$ and $c \in \mathbb{R}^2$	`∃ (lam : ℝ) (c : ℂ)`
$\lambda m_i (q_i - c)$	`((lam * m i : ℝ) : ℂ) * (q i - c)`
$\sum_{j \neq i} m_i m_j (q_i - q_j)/\lvert q_i - q_j\rvert^3$	`∑ j ∈ univ.erase i, ((m i * m j / ‖q i - q j‖ ^ 3 : ℝ) : ℂ) * (q i - q j)`
for $i = 1, \dots, 5$	`∀ i`
q is similar to q'	`Similar q' q`: $q = aq' + b$ or $q = a\bar{q}' + b$ for some $a, b \in \mathbb{C}$, $a \neq 0$
finitely many central configurations up to similarity	`FinitelyManyCC m`: a finite set `F` of configurations such that each `q` with `IsCC m q` is similar to a member of `F`
for every m	`MainTheorem`: `∀ m : Masses`, under the hypothesis above

2.3. **Conventions.** Apart from conventions, the two statements agree. We discuss each convention in turn.

(a) *Indices.* Body k of [24] is `k-1` in `Fin 5`.

(b) *The plane.* The plane is $\mathbb{C}$, with $q = x + iy$. The norm on $\mathbb{C}$ is the Euclidean norm. Real numbers act on $\mathbb{C}$ through the cast $\mathbb{R} \to \mathbb{C}$. Equation (1) in $\mathbb{R}^2$ is therefore the equation of `IsCC` in $\mathbb{C}$.

(c) *The multiplier and the centre.* As in (1), $\lambda \in \mathbb{R}$ and $c \in \mathbb{C}$ are existentially quantified, with no further condition. For positive masses the equation forces $\lambda > 0$ and makes c the centre of mass. We prove $\lambda \neq 0$ (`lam_ne_zero_of_isCC`, `Theory/CorU/SlicePoint.lean`) and $\lambda > 0$ (`lam_pos`, `Core/E3/Norm.lean`), as in the proof of [24, Lemma 2.1], but the statement does not need these facts.

(d) *Collisions.* Division in Lean is total, with $x/0 = 0$. Since `IsCC` requires `CollisionFree q`, the denominators $\|q_i - q_j\|^3$ are nonzero and this convention never applies.

(e) *Reflections.* `Similar` admits all plane similarities, $z \mapsto az + b$ and $z \mapsto a\bar{z} + b$ with $a \neq 0$. If `SimilarOP` replaces `Similar` in `FinitelyManyCC`, only orientation-preserving similarities are admitted. This variant is formally the stronger statement: it implies the formal one at once. Conversely, suppose that a finite set F represents all central configurations up to similarity. Then F together with the complex conjugates of its members represents them up to orientation-preserving similarity, since $a\bar{q}' + b = aq'' + b$ with $q'' = \bar{q}'$. We have not formalized this one-line argument. Thus the formal theorem is the version with reflections, from which the version without them follows by this argument.

(f) *Representatives.* `FinitelyManyCC` asks for a finite set of representatives, which need not be central configurations themselves. Since similarity is an equivalence relation, such a set exists if and only if the central configurations fall into finitely many similarity classes: distinct classes are similar to distinct members of the set.

(g) *Masses.* The only hypothesis on m is $m_i > 0$ for all i, as in Theorem 1.1, and the statement does not normalize, order or scale the masses. Internally, the proof passes to sorted masses as in [24, Lemma 8.1], since settledness and finiteness are invariant under scaling and relabelling (`Settled_invariant_holds` and `Lem_slice_c_holds`, `Core/Reduction.lean`).

2.4. **The trusted base.** To accept Theorem 1.1 from the formal proof, a reader has to accept three things.

(i) The definitions of Listing 1, read with Mathlib's definitions of $\mathbb{R}$, $\mathbb{C}$, the norm of $\mathbb{C}$, complex conjugation, finite sets and finite sums, express Theorem 1.1. This is the comparison above.

(ii) The axioms `propext`, `Classical.choice` and `Quot.sound`, the standard axioms of Lean and Mathlib, are acceptable. For `CC5Full.Core.Compose.MainTheorem_holds`, the command `#print axioms` lists exactly these.

(iii) A Lean kernel checks proofs correctly. The development is built with the toolchain `v4.35.0-rc3`, a release candidate. Lean's own kernel uses the GMP library for arithmetic on natural-number literals. A declaration is checked when its module is built (§7). No file sets the option `debug.skipKernelTC`, which would skip this check (Table 5). Since Lean does not check imported modules again and accepts modules built on other machines by their build traces, this route also trusts the build products. From exports of these products, Lean's kernel checks the closure with the certificate theorems assumed, while two independent kernels check it in full. That route trusts the exporter and the scripts that join the parts instead (§6).

Beyond these points and the infrastructure of item (iii), nothing needs to be trusted. Indeed, the elaborator and the tactics only produce terms for the kernel to check, and Mathlib's proofs are checked in the same way. All other definitions in the project, including the files that state the intermediate results of [24], belong to the proof: if one of them did not mean what its docstring says, `MainTheorem_holds` would still prove `MainTheorem`. Docstrings and names are not checked by the kernel. Finally, the certificate generators and the computer algebra systems behind them only produce data, and the kernel checks those too.

2.5. **What the formal proof does not state.** Theorems 10.1 and 10.2 of [24] count 306 and 294 central configurations up to orientation-preserving similarity on the two rays (183 and 177 up to similarity). They also state that these central configurations are nondegenerate, count them by shape, and give the numbers 612 and 588 of positive normalized central configurations. Theorem 10.7 states the same, with the numbers of Theorem 10.1, at the point $P_1 = (9, 36, 36, 36, 4)$. These statements enter the formal proof only through their finiteness. At each of the three mass points and in each of its five search regions, we prove the exact number of chart points of normalized central configurations in the search box. For instance, `CC5Full.Core.E3.Run.A.count0`

gives 168 (`Core/E3/Run/A/Final.lean`). These numbers add up to 306, 294 and 306, but we do not identify them formally with the numbers of central configurations up to orientation-preserving similarity. Besides these statements and the argument of §2.3(e), the formal proof leaves out the cross-checks of [24, §12.3]. It also bypasses the statements replaced by other arguments in §3.4: Propositions 5.34 and 5.35, the decomposition in the proof of Proposition 8.7(a), and the results of [1] on singular sequences, none of which the formal proof of Theorem 1.1 needs.

3. From the informal to the formal proof

3.1. **The informal proof.** We recall the argument of [24] in the terms used in the formalization. Normalized central configurations, the solutions of (1) with $\lambda = 1$ and $c = 0$, correspond to zeros of a polynomial system in $\mathbb{C}^{18}$. The coordinates are $u_1, \dots, u_4$ and $v_1, \dots, v_4$ for the positions in the two colours $z = x + iy$ and $w = x - iy$, and $p_{kl} = r_{kl}^{-1}$ for the inverse distances, and a central configuration gives a zero with $v = \bar{u}$ and $p > 0$. On this system the potential $U = \sum_{k<l} m_k m_l p_{kl}$ agrees with a quadratic polynomial I in the positions, the polynomial form of the moment of inertia [24, Lemma 2.4(a)]. A slice Y_m removes the rotations [24, §2.2]. If the masses m have infinitely many central configurations up to similarity, then Y_m contains an irreducible curve on which U is not identically zero and some mutual distance is not constant [24, Corollary 2.5]. A mass vector is *settled* if there is no such *nontrivial* curve. The vectors $\rho(P) = (-\operatorname{ord}_P p_{kl})_{k<l}$ at the places at infinity P of a nontrivial curve sum to zero [24, §2.4]. A place is *singular* if the product of the norms of two vectors $\mathcal{Z}$ and $\mathcal{W}$, formed from the coordinates, has a pole there. At a singular place the type $x(P)$, a positive multiple of $\rho(P)$, lies in one of finitely many polyhedral *pieces*. Which pieces can occur depends on polynomial conditions on the masses, the *labels* [24, Theorem 3.12]. For example, with $x_i = m_i^{-1/2}$, the label $\mathsf{E}(abc)$ says that one of x_a, x_b, x_c is the sum of the other two. The label $\mathsf{H}(ij|kl)$ says that $m_i m_j = m_k m_l$, and in label sets it also requires a further condition, Tri, for each of the pairs ij and kl [24, §3.4]. If staged linear functionals separate the pieces present at m, then the types cannot sum to zero, so m is settled [24, Lemma 4.1, Proposition 4.4]. By exact computer algebra and 1840 such certificates, Theorem 8.2 of [24] shows the following for a positive sorted mass vector: its label set is certified, or it contains one of nine *curve sets*, or the vector is a positive multiple of one of two exceptional points. These points are $\mathbf{q}_*$, the only mass vector with $0 < m_1 < \cdots < m_5 = 1$ at which the labels $\mathsf{E}(125)$ and $\mathsf{H}(25|34)$ hold, and $\mathbf{p}_* = (\frac{1}{16}, \frac{1}{9}, \frac{1}{4}, 1, 1)$. At positive masses, Sections 5–7 of [24] exclude the pieces that the certificates cannot handle. [24] treats the curve sets in Section 9, counts the central configurations on the special rays and at P_1 in Section 10, and assembles the proof in Section 11.

3.2. **The interface.** We organize the development around a fixed interface. The statement files `CC5Full/Basic.lean`, `System.lean`, `Places.lean`, `Expansion.lean`, `Diagrams.lean`, `Labels.lean`, `Pieces.lean`, `Germs.lean` and `Interface.lean`, 2717 lines in all, contain the definitions, and the intermediate results stated as propositions, with nothing assumed. The proposition `Interface` is the conjunction of three groups. First, `TheoryForCore` collects the statements about places of curves used by the computations: balancing, Theorems 3.9, 3.12, 3.13 and 3.15 with their variants, the validity of the diagram-5 hulls (Remark 4.5), the exclusion of diagrams 9 and 15 and of the `d14b` pieces of diagram 14 [24, §5], Corollary 2.5 with the invariance of settledness under scaling and relabelling, the meanings of the label E and of the condition Tri, and the tools for expansions at places, among them the formal implicit function theorem and an obstruction lemma. Second, `TheoryForAssembly` consists of Corollary 2.5 and finiteness on the special rays and at P_1. Third, `CoreStatements` collects Lemma 4.1 with the soundness of the exact check of Definition 4.2, Propositions 4.4, 6.7 and 7.1, Lemma 8.1, Theorem 8.2 and Corollary 8.3, the last in the form "settled or a counted point". Between them, the theory part

LISTING 2. The assembly of [24, §11] (`Core/Compose/MainAssembly.lean`, line 18) and the four statements it uses (`System.lean` and `Interface.lean`). The conjunct `hC.2.2.2.2.2.2.2` of `CoreStatements` is `Cor_S`.

```
def Cor_U : Prop := ∀ m : Masses, (∀ i, 0 < m i) → Settled m → FinitelyManyCC m
def Cor_S : Prop :=
  ∀ m : Masses, (∀ i, 0 < m i) → ¬ OnSpecialRay m → Settled m ∨ CountedPoint m
def Finite_specialRays : Prop := ∀ m : Masses, OnSpecialRay m → FinitelyManyCC m
def Finite_P1 : Prop := ∀ m : Masses, IsMultipleOfPerm P1pt m → FinitelyManyCC m

theorem MainTheorem_of_interface_holds : MainTheorem_of_interface := by
  rintro ⟨-, ⟨hU, hRays, hP1⟩, hC⟩ m hm
  by_cases hs : OnSpecialRay m
  · exact hRays m hs
  · rcases hC.2.2.2.2.2.2.2 m hm hs with hset | hcp
    · exact hU m hm hset
    · exact hP1 m hcp
```

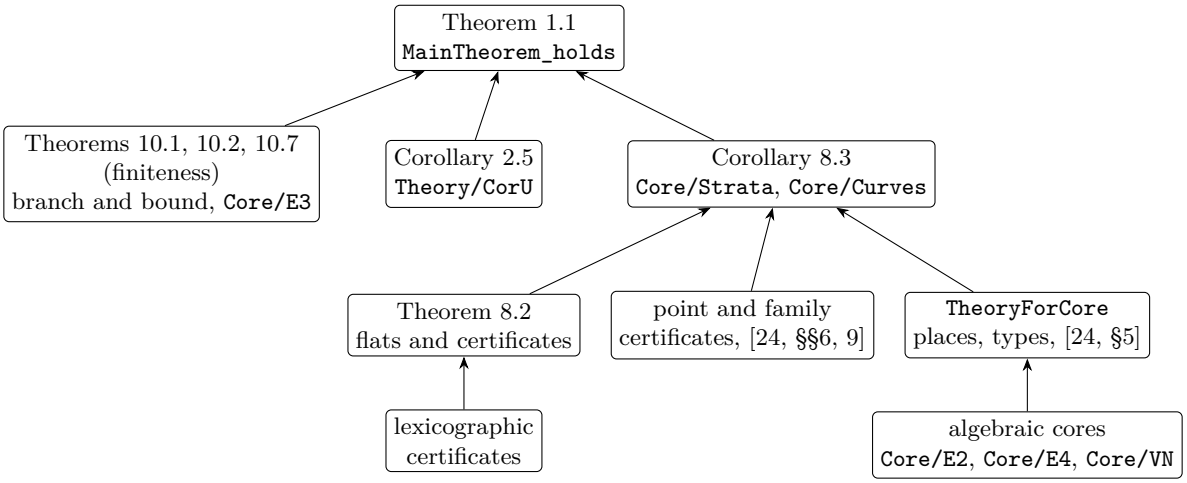


FIGURE 1. The main dependencies of the formal proof. An arrow points from a result to a result that uses it.

(`CC5Full/Theory/`, 261 files, 65,974 lines) and the computational core (`CC5Full/Core/`) prove these statements about the same definitions. We prove the interface in full as `Interface_holds` (`Core/Compose/Main.lean`, line 40).

The step from the interface to Theorem 1.1 is the argument of [24, §11]. We show it in full in Listing 2. A mass vector on a special ray has finitely many central configurations by the count. Otherwise Corollary 8.3 makes it settled or a multiple of a permutation of P_1. In the first case Corollary 2.5 gives finiteness, and in the second case the count at P_1 does. The main dependencies are shown in Figure 1.

3.3. **Informal and formal statements.** In Table 2 we link the main results of [24] to the formal statements and to the theorems that prove them. Most formal statements are propositions defined in the statement files. The theorem that proves a proposition `Foo` is usually named `Foo_holds`. Docstrings in the Lean files cite these results by number and by LaTeX label.

3.4. **Departures from the informal proof.** We follow [24], with the following differences. In (2)–(5) we prove its statement by another route. In (6) and (7) the formal statement is weaker

TABLE 2. Results of [24] and the theorems that prove them. Lean names are relative to the namespace `CC5Full` or `CC5Full.Core`. Paths are relative to `CC5Full/`. The theorems in `Core/Compose` combine theorems of the two parts.

[24]	formal statement	proof (file, line)
Theorem 1.1	`MainTheorem`	`Compose.MainTheorem_holds` (`Core/Compose/Main.lean`, 48)
§11, proof of Theorem 1.1	`MainTheorem_of_interface`	`Compose.MainTheorem_of_interface_holds` (`Core/Compose/MainAssembly.lean`, 18)
Corollary 2.5	`Cor_U`	`Theory.Cor_U_holds` (`Theory/CorU/CorU.lean`, 24)
Lemma 2.4(b)	`Lem_U_b`	`Theory.Lem_U_b_holds` (`Theory/ConstU.lean`, 58)
balancing (2.3), (3.1)	`Balancing`, `BalancingTypes`	`Theory.Balancing_holds`, `Theory.BalancingTypes_holds` (`Theory/FunctionField/Divisor.lean`, 255, 340)
classification of [1, §7]	`Classification`	`Theory.Types.Classification_holds` (`Theory/Types/Residual.lean`, 626)
Proposition 3.5 (Table 1)	`TypesTable`	`Theory.Types.TypesTable_holds` (`Theory/Types/Table.lean`, 32)
Theorem 3.12	`Thm_types`	`Compose.Thm_types_holds` (`Core/Compose/Theory.lean`, 59)
Lemma 4.1	`LexLemma`	`Sep.LexLemma_holds` (`Core/Sep/LexLemma.lean`, 44)
Corollary 5.19	`Cor_nod15`	`Theory.Cor_nod15_holds` (`Theory/D15/Nod15.lean`, 22)
Theorem 5.21	`Thm_D14`	`Compose.Thm_D14_holds` (`Core/Compose/Theory.lean`, 54)
Theorem 5.37	`Thm_D9`	`Theory.Types.Thm_D9_holds` (`Theory/Types/D9.lean`, 15)
Theorem 6.2 (for K14)	`Thm_KA`	`K14.Thm_KA_holds` (`Core/K14/Torus.lean`, 172)
Proposition 7.1	`Prop_nod2`	`Compose.Prop_nod2_holds` (`Core/Compose/Theory.lean`, 48)
Lemma 8.1	`Lem_R`	`Reduction.Lem_R_holds` (`Core/Reduction.lean`, 635)
Theorem 8.2	`Thm_S`	`Strata.Thm_S_holds` (`Core/Strata/Final.lean`, 20)
Corollary 8.3 (settledness part)	`Cor_S`	a conjunct of `Compose.CoreStatements_holds` (`Core/Compose/Assembly.lean`, 39)
Proposition 9.6	`Prop_d14a`	`Compose.Prop_d14a_holds` (`Core/Compose/D14a.lean`, 42)
Theorems 10.1, 10.2 (finiteness)	`Finite_specialRays`	`E3.Finite_specialRays_holds` (`Core/E3/Run/Final.lean`, 23)
Theorem 10.7 (finiteness)	`Finite_P1`	`E3.Finite_P1_holds` (`Core/E3/Run/Final.lean`, 28)
Theorem B.1 and Remark 4.5	`V2_valid`, `VN_valid`	`Compose.V2_valid_holds`, `Compose.VN_valid_holds` (`Core/Compose/Theory.lean`, 66, 70)

than the informal one but suffices for the proof, and in (1) and (8) it differs from the informal one, as described there.

(1) *The inputs from* [1]. In [24], the rules on limit diagrams, the classification into sixteen diagrams [1, §7, Fig. 11] and the mass conditions of [1, §8] are taken from [1], all for singular sequences. We prove the parts of these inputs that the proof needs at singular places of curves (§4.3). Among these results is the reduction of [1, pp. 577–578] to the system of [1, §5.4] (`FigData.kite_core`, `Theory/Types/LabelR.lean`), which [24] cites.

(2) *Corollary 2.5.* We choose the curve so that U is not identically zero on it. The proof then does not use Lemma 2.4(b) (§4.2).

(3) *Theorem 5.21.* At a place whose type lies in a piece `d14b`, a parameter $\sigma \in (1, 2)$ fixes the type. Our proof does not use Proposition 5.11, and it replaces Propositions 5.34 and 5.35 by

two similar arguments, one for $\sigma < 5/3$ (`Theory/D14/CaseLow.lean`) and one for $\sigma > 11/7$ (`Theory/D14/CaseHigh.lean`). Together these ranges cover the range $(1, 2)$ of σ, so the resonant values of σ play no role. We prove Proposition 5.11 itself, which excludes diagram 15 through Corollary 5.19, by the second proof in the remarks after it: a resultant and the positivity of a polynomial P_{11} (`Theory/CC5Alg/`).

(4) *Relation (R2).* Relation (R2) of [24, §3.3] is one of three relations that cut out a smaller hull V_N of diagram 5. We derive it from one polynomial identity and 36 factorizations (the norm over the signs of a branch, `Theory/VN/`), instead of 36 eliminations by standard bases. These factorizations are checked by the kernel (§5.4).

(5) *The zero-dimensional flats* of [24, Proposition 8.6(d)] need no prime decomposition and no real root isolation. For 75 of the 77 flats, an element of the ideal of the flat is positive on the stratum, by a positivity certificate (§5.2). The other two flats contain $\mathbf{q}_*$ and $\mathbf{p}_*$, one each. We treat the flat of $\mathbf{p}_*$ by membership certificates and that of $\mathbf{q}_*$ by the definition of $\mathbf{q}_*$. The certificates at $\mathbf{q}_*$ of Proposition 8.7(b) and (c), however, are checked at an explicit point given by isolating intervals. In the proof of Corollary 8.3 we show that the point defined by the labels is this point. For this we use certificates for 630 auxiliary polynomial systems, the branches of sixteen label sets (`Core/ResidualCert/`). Here 503 systems contain a nonzero integer, so they have no zero. We treat the other 127 by derivations with exact real root isolation. To identify the point, the informal proof instead uses a prime ideal with 48 zeros [24, Proposition 8.7(a)].

(6) *Lemma 10.3(b).* We use the gaps between the projections of the bodies on a body of largest norm instead of the components of the graph $r_{ij} \le \varepsilon$, with the same constants (`gap_lemma`, `Core/E3/Bounds.lean`). The bound holds with $\le$ in place of $<$ (`sq_le_bound`).

(7) *The branch and bound of* [24, §10]. Where a test of the search program of [24] fails in the kernel's arithmetic, the checked tree uses another test, bisections or a local search instead (§5.3). The counts are unchanged. The formal proof proves them as numbers of chart points, not as the statements of [24, §10] about central configurations, and uses only their finiteness (§2.5).

(8) *The pieces of diagrams* 2 *and* 9. In the statement files, the label set $S(m)$ of Theorem 3.12 also contains labels not used in [24], L, $\mathsf{D2K}$, $\mathsf{D2P}$ and N (`CC5Full/Labels.lean`). The piece $\mathsf{d2}[T]$ is present only under $\mathsf{L}(T)$, $\mathsf{D2K}(T)$ or $\mathsf{D2P}(T)$, and $\mathsf{d9}(T; ax, by)$ only under $\mathsf{N}(T; ax, by)$, whereas [24, Table 3] lists both pieces as always present. The formal Theorem 3.12 is therefore stronger than the informal one, and its proof uses Proposition 7.1 and Theorem 5.37, which exclude diagrams 2 and 9 at all positive masses. In [24] these two results remove the same pieces after Theorem 3.12.

3.5. **Generated modules.** We generate most certificates by programs outside Lean, from the data of [24] and its supplementary material, and write them as Lean modules. Including Lean's and Mathlib's declarations, the main theorem depends on 2,831,937 declarations in 90,536 modules. In the library, that is, `CC5Full.lean` and the Lean files of `CC5Full/` outside its `scripts` directories, we store 2,715 files with 420,540 lines. Generators wrote about 2,170 of these files, nearly all of them certificate data; the other files, about 540 with about 129,000 lines, hold the statements, the theory, the checkers and their soundness proofs. Four further families of generated modules used by the proof, with 85,756 files and 3.0 GB of Lean source, are too large to store in the repository (Table 3). For each of them, a script in the repository regenerates the files byte for byte and checks them against a manifest of SHA-256 hashes. To regenerate the generic face of Theorem 8.2 and the 630 auxiliary systems of §3.4(5) one needs Singular 4.4.1 [7], and the generic face also needs data from the supplementary material. For the branch and bound, a C++ program writes the search tree of [24, §10] with one code per node. An exact Python

TABLE 3. The generated families not stored in the repository. Scripts are under `CC5Full/Core/scripts/`.

family (directory)	files	bytes	regenerated by
branch and bound of [24, §10] (`Core/E3/Run`)	83,454	1,560,918,323	`e3/run/regen.sh`
generic face 1|2|3|4|5 of Theorem 8.2 (`Core/Strata/Big/F1_2_3_4_5`)	324	1,128,965,013	`strata/gen_big.py`
the 630 auxiliary systems of §3.4(5) (`Core/ResidualCert/All`)	630	230,621,594	`residual/gen_unit.py`, `gen_all.py`
core of Proposition 9.6 (`Core/D14a/B`)	1,348	83,578,754	`d14a/`

LISTING 3. Curves, nontrivial curves, settledness (`System.lean`) and places (`Places.lean`). `Poly` is `MvPolynomial Coord ℂ`, the polynomials in the eighteen coordinates. `r2P e` is $r_{kl}^2 = u_{kl}v_{kl}$, and `Upoly m` is the potential U.

```
structure Curve (m : Masses) where
  ideal : Ideal Poly
  isPrime : ideal.IsPrime
  le_ideal : idealY m ≤ ideal
  dim_one : ringKrullDim (Poly ⧸ ideal) = 1
def Curve.Nontrivial {m : Masses} (C : Curve m) : Prop :=
  (∃ e, ¬ C.IsConst (r2P e)) ∧ Upoly m ∉ C.ideal
def Settled (m : Masses) : Prop := ∀ C : Curve m, ¬ C.Nontrivial
structure Place (C : Curve m) where
  ord : AddValuation C.FunctionField (WithTop ℤ)
  ord_const : ∀ c : ℂ, c ≠ 0 → ord (C.fn (MvPolynomial.C c)) = 0
  exists_ord_eq_one : ∃ f : C.FunctionField, ord f = 1
```

mirror of the kernel's arithmetic then replays the tree (§5.3). The four families are archived in the supplementary material, with the hashes listed in §9. Since the kernel checks every module (§2.4), a wrong generator can only produce a module that fails to build, or a certificate for a statement that the assembly cannot use.

4. CURVES, PLACES AND THE INPUTS FROM ALBOUY AND KALOSHIN

The argument of [1] studies *singular sequences*: sequences of normalized central configurations along which some mutual distances degenerate, and how the dominant terms of the equations balance along them. Although [24] works with places of curves, it takes the classification of limit diagrams from [1]. We work only with places; each result of [1] that the proof uses is proved at places. Mathlib has the algebra this needs (valuations, Dedekind domains, Noether normalization, ramification indices, Hahn series), but not the theory of places of curves. In this section we describe how we build that theory and how we use it.

4.1. **Curves and places without a projective model.** We show the definitions in Listing 3. Let $\mathbb{C}[u, v, p]$ be the ring of polynomials in the eighteen coordinates. A curve in Y_m is a prime ideal of $\mathbb{C}[u, v, p]$ that contains the ideal of Y_m and whose quotient has Krull dimension one. By the Nullstellensatz these are the irreducible closed curves of $\mathbb{C}^{18}$ inside Y_m. The function field of a curve is the fraction field of its coordinate ring. A place is a discrete valuation of the function field with values in $\mathbb{Z}$, trivial on the nonzero constants and taking the value 1. Curves and places do not occur in the formal statement of Theorem 1.1. They enter through Corollary 2.5, which turns settledness, a statement about all curves in Y_m, into finiteness.

The proof in [24] uses the smooth projective model of a curve, its places at infinity, and the fact that a nonconstant function has as many zeros as poles. We prove the facts we need directly from the definitions (`Theory/FunctionField/`).

- By Noether normalization the coordinate ring of a curve is integral over $\mathbb{C}[t_0]$ for some transcendental t_0 (`curve_noether`), and so the function field has transcendence degree one (`trdeg_eq_one`).
- The integral closure of $\mathbb{C}[t_0]$ in the function field is a Dedekind domain. The maximal ideals of this closure give places centred on the curve, as adic valuations (`exists_place_centred`).
- The residue field of each place is $\mathbb{C}$ (`exists_residue_place`).
- For a transcendental function x, the places with $\operatorname{ord}_P x > 0$ are the primes above (x) in the integral closure of $\mathbb{C}[x]$, and their residue degrees are one. Thus the identity $\sum e_i f_i = [K : \mathbb{C}(x)]$ of Mathlib gives $\sum_{\operatorname{ord}_P x>0} \operatorname{ord}_P x = [K : \mathbb{C}(x)]$ (`degree_pos_places`). Applied to x and to x^{-1}, it shows that a nonconstant function has finitely many zeros and poles, and that its orders sum to zero. Balancing follows (`Balancing_holds`).
- At a place with uniformizer t, expanding functions digit by digit embeds the function field into $\mathbb{C}((t))$ as a ring and turns ord_P into the t-adic order (`Expansion_exists_holds`).

In [24, §5] the expansions have rational exponents, after the substitution $t = \tau^{4/d}$. We model them in Hahn series HahnSeries $\mathbb{Q}$ $(\mathbb{C}[[\delta]])$ over a ring of formal power series in the deviations δ from the leading coefficients (`Expansion.lean`). For the weighted systems of [24, §5] we prove a formal implicit function theorem (`FormalIFT_holds`, `Theory/FormalIFT/`).

4.2. **Two arguments without geometry.** Two steps of [24] use geometric facts about curves. We replace them by algebra.

Corollary 2.5. Suppose that m has infinitely many central configurations up to similarity. Then the set $T = Y_m \cap \{U \neq 0\}$ is infinite (`Theory/CorU/SlicePoint.lean`). Since T is covered by the zero sets of the finitely many minimal primes of its vanishing ideal, one of them contains infinitely many points of T. This minimal prime contains the ideal of Y_m, it does not contain U, and it is not maximal, since a maximal ideal has only one zero. By Noether normalization, lying over and the fact that integral extensions of $\mathbb{C}[X]$ have dimension at most one, some prime of dimension one above it still does not contain U (`Theory/CorU/Algebra.lean`). This prime is a curve on which U is not identically zero. If all squared distances were constant on it, its zeros would lie in a fibre of the map $(r_{kl}^2)_{k<l}$ on Y_m, which is finite (`r2_fibre_finite`). A prime of dimension one, however, has infinitely many zeros, so the curve is nontrivial. Since U is not identically zero on the curve by construction, the proof needs no statement that U is constant on curves.

Lemma 2.4(b), the constancy of U on each curve, is still needed elsewhere, for instance for the energy method of [24, §5] and for Lemma 3.3. We take a place centred on the curve and an expansion ι at it. All coordinate functions have nonnegative order there, so ι maps them to power series. The derivative identity of [24, Lemma 2.3] then gives $\frac{d}{dt}\iota(2U + I) = 0$. Since ι is injective and $U = I$ on the curve, U is constant (`Theory/ConstU.lean`). The proof uses neither smooth points nor connectedness.

4.3. **Orders at places and the classification of diagrams.** At a singular place, the diagram of [1] records which components of two vectors $\mathcal{Z}$ and $\mathcal{W}$ have the largest order of growth. Our definition (`Place.diagram`, `Places.lean`) reads it off the valuations, as in [24, Lemma 3.1]. In [1, §§7–8] the arguments compare the sizes of terms along singular sequences. At a place they become statements about orders. Our basic tool is then the ultrametric inequality: if $f + g + k$ vanishes on the curve, the minimum of $\operatorname{ord} f$, $\operatorname{ord} g$ and $\operatorname{ord} k$ is attained at least twice

(`min2₃W_of_mem`, `Theory/D5Place/Min2.lean`). Then the decision procedure `omega` closes the resulting inequalities between integers.

In [1, §7], a case analysis based on the rules of [1, §3] yields the sixteen diagrams of Fig. 11. We prove each rule at places (`Theory/Types/Rules.lean`) and then enumerate all diagrams in the kernel (`Theory/Types/Enum.lean`). Here a diagram is coded by a number below 4^{10} for the strokes on the ten pairs and by two five-bit masks for the circles. A depth-first search over the pairs prunes as soon as a triangle or a 4-cycle violates a rule. It evaluates the rules through tables stored as single natural numbers and checked once by the kernel against their definitions. In all, the search lists 1376 labelled diagrams that satisfy the rules. A certificate, checked by the kernel in 16 chunks, assigns them to 24 classes up to relabelling and exchange of the two colours (`Theory/Types/EnumCheck.lean`): the sixteen diagrams of [1, Fig. 11] and eight others, which [1] excludes in the course of its classification (their Figs. 9 and 10, Prop. 4 and §§7.5–7.6). We prove by induction on the search that it lists all diagrams that satisfy the rules (`mem_survFrom`, `Theory/Types/EnumSound.lean`). Then we exclude these eight classes at places by corresponding arguments: that of [1, Prop. 4], the rules, the equations of the bodies, and for one class the fact that U has order zero at the place (`Theory/Types/Residual.lean`). The result is `Classification_holds`: at positive masses, a singular place of a nontrivial curve has one of the sixteen diagrams.

The mass conditions of [1, §8] arise at places in the same way. At a singular place with a given diagram, the leading coefficients of the coordinates satisfy reduced equations, which force the labels. For instance, the leading coefficients of a triangle of bodies are in equilibrium, which gives the label `E` (`Theory/Types/LabelE.lean`, `LabelTri.lean`). Those of a kite satisfy the system of [1, §5.4], which gives the label $\mathsf{R}(ab;s)$: $m_s^2(m_a+m_b)^4 = m_a^3 m_b^3$ (`Theory/Types/LabelR.lean`). The theorem `labelsAt_holds` (`Theory/Types/LabelsAt.lean`) collects these conditions for diagrams 1, 4, 6–8 and 10–13; diagrams 3 and 5 are treated by [24, Theorem 3.9] and by the diagram-5 hulls (§3.2).

4.4. **The energy method.** In [24, §5], diagrams 9 and 15 and the pieces `d14b` of diagram 14 are excluded at positive masses by expanding the potential at a place. The potential is constant along the curve. We follow this route in the Hahn-series model of §4.1, with energy identities order by order (`Theory/Energy/`) and the weighted coordinates of [24, §5]. For diagram 14 we show that a place agrees with a truncation of the formal solution along an exact slice up to an explicit order, by a two-point least-exponent argument ([24, Lemma 5.33]; `follow`, `Theory/D14/Follow.lean`). Then we conclude with the two ranges of σ of §3.4(3). Some of these arguments need explicit algebraic facts, such as the computations in the proofs of [24, Theorems 5.28 and 5.29], and the validity of the hull V_N of diagram 5 needs relation (R2). We state such facts as propositions about explicit polynomial data and prove them by kernel computations in the computational core (`Core/E2/`, `Core/VN/`).

5. Certificates checked by the kernel

5.1. **The pattern.** Nearly all computations of the proof are checked in the same way. We define a checker, a function `check : D → Bool` on a type of data, in Lean. Once and for all, we prove in Lean a soundness theorem `check d = true → P d`, where `P` is a statement about Mathlib's real or complex numbers, polynomials or sets. Each certificate is then a theorem `check d0 = true` for an explicit datum `d0`, proved by `decide +kernel`: the kernel itself evaluates `check d0` to `true`. This is proof by reflection [4, 2]. By contrast, the tactic `native_decide` would compile the checker and run machine code, at the price of an axiom, `Lean.ofReduceBool`, through which the compiler and the run-time system have to be trusted. We do not use it, and a keyword scan of `CC5Full.lean` and `CC5Full/`, generated modules included, enforces this

TABLE 4. The theorems in the closure of `MainTheorem_holds` that are proved by evaluation in the kernel, and the modules that contain them, by family. For the branch and bound, these modules include the tops of the trees of its jobs and of the parts into which we cut them (§5.5), the modules that assemble their counts, and five other modules of `Core/E3`. Directories are relative to `CC5Full/`.

family	certificate and checker	theorems	modules
branch and bound of [24, §10]	one code per search-tree node, replayed in fixed-point interval arithmetic (`checkTree`, `Core/E3/Defs.lean`)	621,675	81,545
Propositions G.1, G.2 and the test in Proposition 9.14	Krawczyk tests and lifting recursions in complex ball arithmetic (`Core/Ball/`, `Core/GPts/`)	31,104	972
Proposition 9.6	complex balls and Lyapunov–Schmidt series at special points of the families T_3 and T_7 (`Core/D14a/`)	13,579	1,157
the 630 auxiliary systems of §3.4(5)	cofactor identities over $\mathbb{Z}$, and derivations with root isolation (`Core/ResidualCert/`)	9,560	630
Theorem 8.2, generic face	membership, positivity and chain certificates for the flats (`Core/Strata/`)	1,022	128
all others	lexicographic certificates (`Core/LexBatch/`), runs of K14 modulo primes (`Core/K14Cert/`), [24, §9] (`Core/Curves/`, `Core/Lem95/`), the cores of Theorems 5.28, 5.29 and 7.3 and of (R2) (`Core/E2/`, `Core/E4/`, `Core/VN/`), the other faces of Theorem 8.2, and the theory	3,172	869
all		680,112	85,301

(`check.sh`). In the closure of `MainTheorem_holds`, 680,112 of our theorems have proofs of the form `of_decide_eq_true`, the term produced by such evaluations. They lie in 85,301 modules.

Evaluation in the kernel is by reduction to weak head normal form. The kernel has fast arithmetic only for natural-number literals: it hands operations such as addition, multiplication, division, comparison and bitwise operations on two literals to the GMP library. Hence a checker is fast when its arithmetic reduces to these operations. We list the families of certificates and their checkers in Table 4. There K14 is the system satisfied by four-body clusters of zero energy [24, §3.7], T_3 and T_7 are one-parameter families of masses [24, §9], and Propositions G.1 and G.2 of [24] exclude diagram 6 with centre 4 and wings $\{1,4,5\}$, $\{2,3,4\}$ at the mass point $(9,36,36,4,9)$ and diagram 3 at the two mass points $(1,1,t^{-2},t^{-2},(1+t)^{-2})$ with $t \in \{t_*, 1/t_*\}$, where $t_* \approx 1.8025$ is the root greater than 1 of $2t^8+t^6-8t^5-6t^4-8t^3+t^2+2$. A test of the same kind for diagram 6, at a point of the family T_9, enters Proposition 9.14. Below we describe the mathematics of two of them, and then the encodings and techniques that they all share.

5.2. **Theorem 8.2: certificates for real algebraic sets.** Theorem 8.2 of [24] concerns all sorted positive mass vectors. A face F of the sorted cone, one of sixteen, prescribes which consecutive sorted masses are equal, and the masses are scaled to $m_5 = 1$. On the stratum Σ_F put $x_i = m_i^{-1/2}$. The values $y_1 > \cdots > y_n$ of x_i on the blocks of equal masses, other than the last block, where $x_i = 1$, identify Σ_F with the open cone $\{y_1 > \cdots > y_n > 1\}$ [24, §8.4]. In these coordinates a label becomes a union of algebraic sets, the zero sets of its branch ideals. On every face, the computation of [24, §8] gives a tree of ideals, called flats, with 2206 flats in total. We replay this tree (`Core/Strata/Descent.lean`) and check three kinds of certificates.

- *Membership*: an identity $\sum_j c_j g_j = d\, r\, k^N$ with $d \neq 0$ in integer polynomials. It shows that r or k vanishes at each common zero of the g_j.
- *Positivity*: an element s of an ideal, possibly after multiplication by a power of $\prod_j y_j \prod_j (y_j - y_{j+1})\,(y_n - 1)$, such that $s \circ \varphi$ is a nonzero polynomial with coefficients of one sign, where $\varphi(z)_j = 1 + z_j + \cdots + z_n$. Since φ maps the open positive orthant onto the cone, s has no zero on the stratum (`posOK_sound`, `Core/Strata/Pos.lean`). In the kernel, $s \circ \varphi$ is a composition of shifts $x_k \mapsto x_k + x_{k+1}$, with binomial rows by Pascal's rule.
- *Chains*: residual chains of ideals, which show that a zero of an ideal in the cone lies on one of its children.

A descent by strong induction on the number of labels that are not attached to a flat then gives Theorem 8.2 face by face (`face_sound`). Positivity certificates of degree at most 8, which we find by linear programming, settle 75 of the 77 zero-dimensional flats of case (d) of [24, Proposition 8.6]. In [24] these flats are treated by prime decompositions, filters and, for four of them, real root isolation (§3.4(5)). No positivity certificate exists, of course, for a flat with a zero in the open cone, such as the flat that contains $\mathbf{q}_*$. For that flat we use the definition of $\mathbf{q}_*$ in the statement of Theorem 8.2. The certified label sets are those of the 1840 lexicographic certificates of [24], checked by the kernel in 147 declarations (`Core/LexBatch/ThmS/`).

5.3. **The branch and bound: replaying a search.** Theorems 10.1, 10.2 and 10.7 of [24] count central configurations at the mass points $(1, 4, 4, 4, 4)$, $(1, 1, 4, 4, 4)$ and P_1. By scaling and relabelling [24, Lemma 2.2(c)], these three points cover the two rays and the multiples of the permutations of P_1. After a similarity a central configuration is normalized, with $\lambda = 1$ and $c = 0$ in (1). For fixed tie-breaking constants c_i, it lies in region k if body k maximizes $|q_i|^2 + c_i$. In region k we rotate the configuration so that q_k lies on the positive real axis and drop one body, whose position the centre of mass determines. The remaining seven real coordinates form its *chart point*. In region k the chart points lie in a box B_k of $\mathbb{R}^7$ and are zeros of a reduced system of seven equations [24, §10]. A branch and bound tiles each box. Each leaf is either excluded by a test or certified by Krawczyk's test [14, 19] to contain exactly one zero, in its interior. The tests are the region conditions, the virial identity, cluster identities for subsets of bodies, mean-value forms of the equations, and disjointness of a box from its Krawczyk image. The checker rests on two pieces of analysis. The first, `krawczyk_box` (`Core/E3/KrawBox.lean`), is a Krawczyk theorem on rectangular boxes, for maps with partial derivatives along coordinate lines and interval bounds on them: if the Krawczyk image lies in the open box, the map has exactly one zero in the box. The second, `tiling_count` (`Core/E3/Tiling.lean`), counts the zeros of a tiling whose excluded leaves contain none and whose Krawczyk leaves contain one each.

In [24] the search ran in floating point with outward rounding. Our checker, however, uses its own arithmetic (§5.4), so a test that passes in floating point can fail in the kernel. For this reason we record one code per tree node as a hint: bisect along a given coordinate, contract, or apply a given test. The Python mirror of §3.5 replays the tree, checks the hinted tests, and replaces a test that fails by another test (once after a contraction), by bisections or by a local branch and bound. The kernel checks the final tree, so the floating-point search and the replay by the mirror stay outside the trusted base of §2.4. The search of [24] is split into 44 jobs, whose boxes tile the boxes B_k. Altogether the final trees of the jobs have 447,097,359 nodes. For all 44 search jobs, the kernel's count equals the number of zeros in region k recorded in the log of the job, and these logs are part of [25] (Table 5).

5.4. **Encodings for the kernel.** All checkers compute with natural numbers. We choose encodings that turn the kernel's work into GMP operations on large literals.

- *Intervals.* In the branch-and-bound checker, a natural number X stands for the real number $(X - 2^{512})/2^{64}$. Intervals are pairs of such numbers with outward rounding. A magnitude bound certifies that no truncated subtraction underflows (`Core/E3/Defs.lean`). Since `Nat.sqrt` is not among the operations accelerated by the kernel, we compute integer square roots by Newton's iteration and check them by $s^2 \le n < (s+1)^2$. Complex balls in a biased fixed-point format serve the Krawczyk tests of Propositions G.1 and G.2 (`Core/Ball/`).
- *Polynomials.* A sparse polynomial is a list of terms whose exponent vectors are packed into one natural number, so that adding keys multiplies monomials (`Core/Sparse.lean`). A cofactor identity $\sum_j h_j G_j = g$ with millions of products does not fit in one declaration. We split the products by a label, check the intervals of labels in separate declarations, and prove that the parts add up to the identity for any choice of labels (`identity_of_parts`, `Core/K14Cert/Sound.lean`; `Ident.sound_of_parts`, `Core/Strata/Ext.lean`). Thus the labels affect only the cost of the check, not its soundness.
- *Identities by one evaluation.* We check a polynomial identity in $\mathbb{Z}[t, k, d]$ with degree bounds D_1, D_2 by one evaluation at $(2^B, 2^{BD_1}, 2^{BD_1D_2})$, where 2^B exceeds twice a bound on the ℓ^1 norm of the coefficients (`evC_eq_zero_of_check`, `Core/E2/Kron.lean`). Since the 36 factorizations of §3.4(4) are weighted homogeneous, we may set one variable to 1 and use this three-variable test. Together, the 36 identities take 40.3 s of kernel time.
- *Lexicographic certificates.* A lexicographic certificate [24, Definition 4.2] is a list of integer functionals that must be nonnegative and separating on the attained faces of all pieces present for a label set. Our checker `LexCert.checkFast` (`Core/FastLex/`) packs vertex coordinates into 48-bit fields of one natural number and evaluates a functional on many vertices with a few multiplications and masks. The soundness theorem `LexCert.checkFast_sound` reduces it to the exact check `lexCheck` in the statement files. We generate the packed tables of the pieces by evaluation in Lean and check them in the kernel against the definitions in the statement files (`Core/FastLex/Valid.lean`).
- *Rationals.* The rationals of Lean's core library are normalized fractions, whose arithmetic the kernel can evaluate. Two checkers compare rationals by their numerators and denominators (`qeqb`, `Core/QEqb.lean`) rather than through the derived decision procedure for equality; we will explain why in §6.2.

5.5. **Writing checkers for the kernel.** The cost of a check in the kernel depends on how the checker is written. For an interval multiply-add of the branch-and-bound arithmetic we measured 79 μs of kernel time when its case distinctions are written with `Bool.rec`, 117 μs with `cond`, 155 μs with `if`, and 310 μs with integers instead of biased natural numbers. Since the kernel unfolds `Nat.sqrt` and `Nat.log2`, they are slow: 2.1 ms and 1.7 ms on 200-bit numbers. In the enumeration of diagrams we write the functions with the recursors `Nat.rec` and `List.rec` on primitive operations. Functions defined by structural recursion, which go through `brecOn`, are evaluated more slowly.

The cost also depends on the size of a declaration. In the branch and bound, the kernel time per leaf grows from 5.4 ms in a declaration of 81 leaves to 9.7 ms in one of 2000 leaves. We therefore cut the trees of the jobs into parts, and the parts into subtrees of weight at most 2048, eight to a module, where a cheap leaf weighs 1 and a node with a mean-value form, a contraction or a Krawczyk test 12 to 30. Batches of lexicographic certificates stay below about $5 \cdot 10^5$ kernel unfoldings per declaration. For Theorem 8.2 a declaration needs about 80 KB of memory per product of terms in its identities, so the generator cuts the declarations to at most $8 \cdot 10^4$ products each, counted in advance.

A different problem arises when the kernel checks the soundness proofs, in which the functions of the checker are applied to variables. A definitional unfolding of a conditional over a symbolic

argument can send the kernel into the well-founded recursion behind `Nat.div` and `Nat.mod`, where the check ends in deep recursion. We avoid this by rewriting with equation lemmas before such a step.

5.6. **Statements of certificate formats.** The hypotheses of a soundness theorem, which states what a certificate format proves, must exclude degenerate data. To show that these hypotheses are needed, we prove in the development that several soundness statements without them are false. For example, a lifting lemma for polynomial systems modulo a prime needs all polynomials in the system to be nonzero, since a zero polynomial makes one of its hypotheses vacuous (`not_LiftingLemma_noP9`, `Core/K14/Counter.lean`). A certificate that identifies a root of an eliminant with a known point must check that the coordinate of the point is that root, not only that it lies in the isolating box of the root (`not_ZeroCert_sound`, `Core/ResidualCert/Counter.lean`). Similarly, the replay of a tree that defers subtrees to other declarations is sound only if the deferred boxes are nondegenerate (`checkTree_sound_as_requested_is_false`, `Core/E3/Replay.lean`). All statements used in the proof carry these hypotheses.

6. Independent kernels

The certificates rely on evaluation by the kernel far more than ordinary proofs do. We therefore checked the closure of `MainTheorem_holds` again with two independent Lean kernels: nanoda 0.4.17 [20], and lean4lean [5] in export mode, both in the versions shipped with the toolchain. Table 5 collects the results.

6.1. **Method.** Evaluating the certificates dominates the cost of the check, which in one process would take weeks. We therefore split the check into parts that run in parallel. Part A is the closure with the 680,112 certificate theorems replaced by assumptions with the same names and types, an export of 7.0 GB. Part B consists of the certificate theorems with their full closures on the three standard axioms, grouped by modules into 6,731 exports of 54.8 GB in all. We ran part B at an earlier commit. Ten of its exports have closures that differ from those of the released library, and we ran these ten again on the released library. A Python script, the glue, then checks that part A assumes exactly the certificate theorems and the three axioms, that part B proves the certificate theorems with the types assumed in part A, that the declarations shared by the parts are identical, and that the exports of part B agree with each other and declare no other axioms. Here declarations are compared by SHA-256 hashes of a canonical form. A cross-check compares our exports, written by `tools/kernel2/kexport3`, with those written by the toolchain's exporter lean4export [15]. As a control, we changed one certificate in one export to claim `false` instead of `true`. Both kernels rejected the changed export, and the glue reported exactly that declaration.

Both kernels accepted the closure. lean4lean bounds its reduction loops by a configurable fuel and reports running out of fuel as a timeout, not as an error in the proof. With the default fuel it stopped on 36 of the 6,731 exports of part B. With ten times the default it accepted all of them.

6.2. **An issue in nanoda.** With the decision procedure derived by Lean for the structure `Rat`, nanoda 0.4.17 overflows its stack on four certificate theorems of two of our checkers, `IdentData.check` and `Sep.eqCert`, for the curve families of [24, §9]. Evaluating this procedure makes the kernel compare the unevaluated numerators and denominators on the two sides. These are different expressions with the same value. They contain proofs of divisibility facts from the normalization of fractions. When two proofs of propositions that are not definitionally equal meet, Lean's kernel and lean4lean conclude at once that the proofs are not equal. nanoda instead goes on to unfold the proofs, down to the well-founded recursion of `Nat.gcd.induction` on numbers of 20 to 37 digits. A product of two fractions with 22-digit numerators over 2^{100}, compared by `decide +kernel`,

TABLE 5. The checks of the formal proof and their costs. Unless its row says otherwise, a check was run at a commit at which the Lean files of the library are those of the release.

check	result	cost
keyword scan of `CC5Full.lean` and `CC5Full/`, generated modules included	none of `sorry`, `admit`, `axiom`, `native_decide`, `implemented_by`, `extern`, `ofReduceBool`, `unsafe` in 90,136 files	9 s
search for the option `debug.skipKernelTC` in the archives listed in §9, which hold the project at `52858e1` and the four families	no occurrence	–
regeneration of the four families of Table 3 and of the generated modules stored in `Core/GPts`	byte-identical to the hash lists and to the stored modules	22.61 CPU-h for the branch and bound
regeneration, and build of all project modules by Lean's kernel, Mathlib prebuilt, in the first run of §7, at a commit that differs from the release in three Lean files of the library	all targets built	about 863 CPU-h; 2.34 h of wall-clock time
rebuild of these three files and of the 794 modules that import them, in the second run	all targets up to date	2.97 CPU-h for the 748 built on the cluster
closure of `MainTheorem_holds` exported from that build, against the state checked by the kernels	byte-identical, 7,410,727,737 bytes	0.82 CPU-h
kernel counts of the 44 jobs of the branch and bound, against the zeros in region k recorded in their logs in [25]	equal	–
`#print axioms` of `MainTheorem_holds`, and the 29 runs of the axioms scripts	`propext`, `Classical.choice`, `Quot.sound` only	14 s for `#print axioms`
part A, Lean's kernel (`leanchecker`)	accepted	1,493 CPU-s
part A, nanoda	accepted	546 CPU-s
part A, lean4lean	accepted	1,853 CPU-s
part B, nanoda (§6.1)	all 6,731 exports accepted	575.4 CPU-h at an earlier commit
part B, lean4lean (§6.1)	all 6,731 exports accepted, 36 of them with ten times the default fuel	980.2 CPU-h at an earlier commit
glue; cross-check against lean4export	no problem	18 s; 16 s
control: one certificate claims `false`	rejected by both kernels; the glue reports exactly that declaration	145 s

reproduces the overflow in a few lines of core Lean (`CC5Full/Core/scripts/kernel2/diag/`). This is why the two checkers of §5.4 use `qeqb a b := a.num == b.num && a.den == b.den`, with `qeqb_iff`. With this comparison, nanoda accepts all certificate theorems.

7. REPRODUCTION

The supplementary material [25] holds the source of the project at the released commit `52858e1` and the four families of generated modules (§9). In the file `README.md` of the project, the commands are given at three levels: `check.sh`, which scans the sources for forbidden keywords, builds the default target and prints the axioms of 29 theorems; the theory; and Theorem 1.1 with the generated families.

We reproduced the build in two runs from fresh copies of the sources, with Mathlib and the other packages prebuilt, on a main machine with at most 16 Lean processes and on a

cluster of machines. The first run, of a commit that differs from the release in three Lean files of the library, regenerated the families and built all project modules from source. The second run, of the released library, compiled from source only these three files and the 794 modules that import them, and took the build products of the other modules from a separate build of the first run's commit. Each run also executed the axioms scripts (29 runs) and `#print axioms CC5Full.Core.Compose.MainTheorem_holds` from a new file, which printed `[propext, Classical.choice, Quot.sound]`. Table 5 lists the checks with their costs. The file `README_records.md`, which accompanies this paper, gives the commit at which each check ran, the records of the project from which we take our figures, and further details of the two runs and of the deposited archive.

8. Discussion

8.1. **Comparison with other formal proofs.** Several formal proofs check large computations. Gonthier's proof of the four colour theorem in Coq [9] runs its case analysis by reflection in the evaluator of the proof assistant. In the formal proof of the Kepler conjecture [10], which combines HOL Light and Isabelle, the verification of more than 23,000 nonlinear inequalities took about 5000 processor-hours. Immler [13] verified the computer-assisted part of Tucker's proof [27] that the Lorenz attractor exists, Smale's fourteenth problem, with an ODE solver verified in Isabelle/HOL. Proofs of numerical bounds by computation inside a proof assistant include Melquiond's work in Coq [18, 16]. Solovyev and Hales verified the nonlinear inequalities of the Kepler proof with Taylor intervals in HOL Light [22]. In SAT-based proofs in combinatorics, such as that of the Boolean Pythagorean triples theorem [12], an untrusted solver produces a proof that a small checker validates. For the empty hexagon theorem, the encoding was verified in Lean [23]. The uniqueness proof for convex four-body central configurations in [26] is also formalized in Lean, with its branch and bound checked by the kernel, on a much smaller computation.

Like the formal proof of the Kepler conjecture, our development formalizes the mathematical framework of its computations, here the theory of [1] at places, together with the computations. All of its computations are evaluated by the Lean kernel itself, without compiled code or any checker outside the proof assistant, at the cost of a build of about 863 CPU-hours. The certificates are of several kinds at once: exact algebra over $\mathbb{Q}$ and modulo primes, ball and interval arithmetic, linear algebra, and positivity on cones.

8.2. **Concluding remarks.** Two observations may be useful for other proofs by reflection of this size. First, the cost of evaluation by the kernel depends on how a checker is written and on how the certificates are cut into declarations (§5.5). Second, an independent kernel may fail on declarations that Lean's kernel accepts: nanoda failed on four certificate theorems of two of our checkers until these compared rationals by their numerators and denominators (§6.2). Among the points that stay outside the formal proof (§2.5) are the version of Theorem 1.1 without reflections, which follows from the formal one by the argument of §2.3(e), and the identification of the counts of the branch and bound with numbers of central configurations.

9. Availability

The formalization is part of the supplementary material of [24], deposited on Zenodo [25] (`https://doi.org/10.5281/zenodo.23202861`). The archive `cc50-rung2-lean_full.tar.xz` holds `git archive` of commit `52858e1c7e831f226c306d7b3bdf8d02c16f38da` of the project, without build products and without the generated families, and with three docstring phrases reworded in two files. The generated families are in four archives, which unpack at the root of the project:

- `cc50-rung2-lean_full-E3-Run-modules.tar.xz`, SHA-256 `b490708baaffd93208a3ef1c0adb0b1a3ed9869d1c0df43e94233455ac49ac68`;
- `cc50-rung2-lean_full-strata-F1_2_3_4_5-modules.tar.xz`, SHA-256 `ae4af0728b757a85e31c2907e5025686d2d8e57106440efe8c8addbed34fc1e1`;
- `cc50-rung2-lean_full-ResidualCert-All-modules.tar.xz`, SHA-256 `ba1f667699944838e7b959b5f13e5595867e5a8a1dbb1df01c8ab6f42b8a596c`;
- `cc50-rung2-lean_full-D14a-B-modules.tar.xz`, SHA-256 `211cb529c6ab8c122681206e6ead8c69635ff72effc564fe5d6a988124461f1e`.

The toolchain is `leanprover/lean4:v4.35.0-rc3`. The file `lake-manifest.json` pins Mathlib at commit `c55e6e786f49471c72fbddbec5415808896aec1e`.

Acknowledgements

Generative AI tools were used to write code, run analyses and draft text; the author directed the work and takes full responsibility for its content.

Funding

This work received no external funding.

References

[1] A. Albouy and V. Kaloshin, *Finiteness of central configurations of five bodies in the plane*, Ann. of Math. (2) **176** (2012), 535–588.
[2] H. Barendregt and E. Barendsen, *Autarkic computations in formal proofs*, J. Automat. Reason. **28** (2002), 321–336.
[3] A. Bastounis, F. Circelli and A. C. Hansen, *Navier-Stokes lost in translation: Why Lean verification of AI autoformalisation does not guarantee correct natural language proofs*, arXiv:2610.08144 (2026).
[4] S. Boutin, *Using reflection to build efficient and certified decision procedures*, in: Theoretical Aspects of Computer Software (TACS 1997), Lecture Notes in Comput. Sci. **1281**, Springer, 1997, 515–529.
[5] M. Carneiro, *Lean4Lean: Verifying a typechecker for Lean, in Lean*, arXiv:2403.14064v3 (2025).
[6] Z. Chen and Z. Tong, *Finiteness of planar central configurations of five bodies for all positive masses: On Smale's sixth problem*, arXiv:2610.16107 (2026).
[7] W. Decker, G.-M. Greuel, G. Pfister and H. Schönemann, *Singular 4-4-1 — A computer algebra system for polynomial computations*, `https://www.singular.uni-kl.de` (2025).
[8] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in: Automated Deduction (CADE 28), Lecture Notes in Comput. Sci. **12699**, Springer, 2021, 625–635.
[9] G. Gonthier, *Formal proof—the four-color theorem*, Notices Amer. Math. Soc. **55** (2008), 1382–1393.
[10] T. Hales, M. Adams, G. Bauer, T. D. Dang, J. Harrison, L. T. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, Q. T. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, T. H. A. Ta, N. T. Tran, T. D. Trieu, J. Urban, K. Vu and R. Zumkeller, *A formal proof of the Kepler conjecture*, Forum Math. Pi **5** (2017), e2.
[11] M. Hampton and R. Moeckel, *Finiteness of relative equilibria of the four-body problem*, Invent. Math. **163** (2006), 289–312.
[12] M. J. H. Heule, O. Kullmann and V. W. Marek, *Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer*, in: Theory and Applications of Satisfiability Testing (SAT 2016), Lecture Notes in Comput. Sci. **9710**, Springer, 2016, 228–245.
[13] F. Immler, *A verified ODE solver and the Lorenz attractor*, J. Automat. Reason. **61** (2018), 73–111.
[14] R. Krawczyk, *Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken*, Computing **4** (1969), 187–201.
[15] lean4export, an exporter of Lean 4 environments, version 3.1.0, `https://github.com/leanprover/lean4export`.
[16] É. Martin-Dorel and G. Melquiond, *Proving tight bounds on univariate expressions with elementary functions in Coq*, J. Automat. Reason. **57** (2016), 187–217.
[17] The mathlib Community, *The Lean mathematical library*, in: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), ACM, 2020, 367–381.

[18] G. Melquiond, *Proving bounds on real-valued functions with computations*, in: Automated Reasoning (IJCAR 2008), Lecture Notes in Comput. Sci. **5195**, Springer, 2008, 2–17.
[19] R. E. Moore, *A test for existence of solutions to nonlinear systems*, SIAM J. Numer. Anal. **14** (1977), 611–615.
[20] nanoda, a type checker for Lean 4, version 0.4.17, `https://github.com/ammkrn/nanoda_lib`.
[21] S. Smale, *Mathematical problems for the next century*, Math. Intelligencer **20** (1998), no. 2, 7–15.
[22] A. Solovyev and T. C. Hales, *Formal verification of nonlinear inequalities with Taylor interval approximations*, in: NASA Formal Methods (NFM 2013), Lecture Notes in Comput. Sci. **7871**, Springer, 2013, 383–397.
[23] B. Subercaseaux, W. Nawrocki, J. Gallicchio, C. Codel, M. Carneiro and M. J. H. Heule, *Formal verification of the empty hexagon number*, in: 15th International Conference on Interactive Theorem Proving (ITP 2024), LIPIcs **309**, 2024, 35:1–35:19.
[24] T. S. Tomar, *Smale's sixth problem for five bodies in the plane, with a proof formalized in Lean*, version 2, Zenodo (2026), `https://doi.org/10.5281/zenodo.23406010`; version 1: arXiv:2610.13100 (2026), titled *Finiteness of planar central configurations of five bodies for all positive masses*.
[25] T. S. Tomar, *Supplementary material for "Smale's sixth problem for five bodies in the plane, with a proof formalized in Lean"*, Zenodo (2026), `https://doi.org/10.5281/zenodo.23202861`.
[26] T. S. Tomar, *Uniqueness of four-body convex central configurations for all masses*, arXiv:2609.35632 (2026).
[27] W. Tucker, *A rigorous ODE solver and Smale's 14th problem*, Found. Comput. Math. **2** (2002), 53–117.

ORCID: `https://orcid.org/0000-0003-4668-3639`