Faster Verified Real Root Isolation with Descartes' Rule of Signs (Short Paper)

Aeacus Sheng
University of Edinburgh, UK

Wenda Li
University of Edinburgh, UK

Paul B. Jackson
University of Edinburgh, UK

Abstract

Real root isolation is a fundamental subroutine in computer algebra, with applications ranging from algebraic number arithmetic to solving polynomial systems. Modern implementations typically employ subdivision methods based on root counting via Descartes' rule of signs. In contrast, most existing formally verified root isolation procedures rely on Sturm's theorem for root counting, leading to a noticeable gap between practical implementations and formally verified approaches.

We take an initial step toward efficient verified real root isolation by formally verifying two simple algorithms based on Descartes' rule of signs: a classical bisection procedure and a Newton-accelerated variant. In this paper, we describe the algorithms, present formal proofs of termination, soundness, and completeness, and discuss efforts made for code generation. Brief experiments show promising performance improvements over existing formally verified algorithms in Isabelle/HOL.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Isabelle/HOL, Real root isolation, Descartes' rule of signs

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/Aeacu2/Dsc`

1 Introduction

Isolating the real roots of an integer univariate polynomial is a fundamental task in computer algebra, with applications in algorithms ranging from complex root finding [8] to quantifier elimination [7]. Although present in almost every modern computer algebra system [2, 5, 30] and leading SMT solvers like Z3 [1], an implementation of a subdivision algorithm based on Descartes' rule of signs is not yet available in any proof assistant. Existing verified root isolation algorithms have largely relied on Sturm sequences [17, 10, 24, 25], whose core computations are based on polynomial remainder sequences rather than the cheaper coefficient sign variation reasoning when using Descartes' rule of signs. Various empirical tests and theoretical complexity analysis have demonstrated the superiority of Descartes-based algorithms over those based on Sturm sequences [13, 16, 15]. Moreover, Newton iteration has been used to accelerate state-of-the-art Descartes-based algorithms for difficult polynomials, making them even faster [18].

While Descartes-based algorithms have not been developed in interactive theorem provers, key mathematical ingredients are already present in many theorem provers [34, 4, 29, 33]. Prior work in Isabelle/HOL [26] includes the formalisation of Descartes' Rule of Signs [23, 11] and the Theorem of Three Circles [32]. Leveraging these results and a range of existing libraries, we have verified a classical bisection algorithm Dsc and a Newton-accelerated NewDsc. The algorithms are executable via Isabelle's code generator [14] and show strong performance against existing verified algorithms based on Sturm sequences. To the best of

our knowledge, these are the first verified Descartes-based real root isolation algorithms in any theorem prover.

An eventual goal is to produce a verified root-isolation procedure whose performance approaches that of unverified state-of-the-art algorithms and which can replace corresponding procedures in existing Isabelle work [10, 17, 22]. Probably the best performing of these is in Li, Passmore, and Paulson's work [22], where root sets are computed by an external unverified untrusted program and then certified correct in Isabelle. Our approach has the potential to leverage the recent progress on Isabelle-LLVM [20, 12] — which has produced verified code on par with hand-optimised C/C++ — to build a verified yet performant procedure that removes both the external-tool dependency and the kernel-side certification overhead.

The rest of the paper will explain the algorithms, their verification, and code generation in Isabelle. First, Section 2 reviews Descartes' rule of signs and its existing formalisations in Isabelle, leading to the formal definition of the classic bisection algorithm. Its termination, soundness, and completeness proofs are explained in Section 3. Then, we introduce the Newton-accelerated algorithm in Section 4, and discuss challenges in verifying its completeness. Section 5 explains efforts made to generate efficient code and presents experimental data comparing the performance of our algorithms against the best verified alternative. In the end, Section 6 concludes by detailing directions for future work.

2 Descartes' Rule of Signs and the Algorithm Dsc

Before presenting the algorithm Dsc, we briefly describe Descartes' rule of signs. Its proof can be found in Basu, Pollack, and Roy's textbook [3].

2.1 Descartes' Rule and Coefficient Sign Changes

▶ **Theorem 1** (Descartes' rule of signs). *Suppose we have a nonzero real polynomial* $P(x) = \sum_{i=0}^{n} c_i x^i \in \mathbb{R}[x]$. *Let* Var_P *be the number of sign changes in the coefficient list* $(c_n, \ldots, c_0)$ *after removing zeros, and* $\mathrm{N}_{>0}(P)$ *be the number of positive real roots of* P*, counted with multiplicity. Then,* $\mathrm{N}_{>0}(P) \leq \mathrm{Var}_P$ *and* $\mathrm{Var}_P - \mathrm{N}_{>0}(P)$ *is even.*

As an immediate corollary, $\mathrm{N}_{>0}(P) = \mathrm{Var}_P$ when Var_P is 0 or 1. In its classical form, the rule tells us about the number of positive real roots of a polynomial by the number of sign variations in its coefficients. To apply this reasoning to an arbitrary interval (a, b), modern algorithms use a coordinate transformation $x \mapsto \frac{ax+b}{x+1}$, sending $\mathbb{R}_{>0}$ bijectively to (a, b), and consider the coefficient sign changes of the associated polynomial $P_{(a,b)}(x) := (x+1)^n \cdot P\left(\frac{ax+b}{x+1}\right)$. Eberl was the first to prove Descartes' rule of signs in Isabelle [10], while Li independently formalised the rule and the derived $\mathrm{Var}_{P_{(a,b)}}$ to bound the number of roots in any interval (a, b) [21, 23].

Polynomials of degree at most p form a vector space with, as one basis, the monomials $x^0, \ldots, x^p$. Bernstein polynomials of degree p with real-valued parameters a, b serve as an alternate basis [3], and it can be convenient to express polynomials of degree at most p in one of these bases. Thompson and Li [32] defined `Bernstein_changes p a b P` as the number of sign changes of the coefficients of a polynomial P when expressed in the Bernstein basis of degree p for parameters a, b. Similar to the case with the monomial basis, they proved results that relate `Bernstein_changes p a b P` to the number of roots in the interval (a, b). Moreover, we proved that `Bernstein_changes p a b P` is equal to Li's formalisation of $\mathrm{Var}_{P_{(a,b)}}$ when `p = degree P`, bridging existing work.

2.2 The Classical Bisection Algorithm

Using Descartes' rule of signs, the classical real root isolation algorithm Dsc recursively bisects intervals until `Bernstein_changes p a b P` is 0 or 1, in addition to checking whether the midpoint is a root. Our formalisation is straightforward in Isabelle:

```
function (domintros) dsc :: "nat ⇒ real ⇒ real ⇒ real poly ⇒ (real ×
    real) list"
where
  "dsc p a b P =
    (let v = Bernstein_changes p a b P in
     if v = 0 then []
     else if v = 1 then [(a, b)]
     else
       (let m = (a + b) / 2 in
          (if poly P m = 0 then [(m, m)] else []) @
          dsc p a m P @ dsc p m b P))"
```

The main inputs `a b P` correspond to the endpoints of the interval (a, b) and the polynomial P. A list of isolating intervals (some are roots in the form `(m, m)`) is produced as the algorithm runs, and it will terminate as long as the polynomial is square-free. Future work can also wrap this function with formalisations of polynomial factorization [31, 9] and Cauchy bound computation to isolate all real roots for non-square-free polynomials, as in [17].

3 Verifying Dsc

3.1 Termination

Dsc terminates if and only if repeated bisection is guaranteed to eventually reach intervals on which `Bernstein_changes` returns 0 or 1. The key bridge from interval width to this "0/1" situation is the Theorem of Three Circles, proved in Isabelle/HOL by Thompson and Li [32]. The theorem considers a polynomial P and an open interval (a, b), and defines three overlapping discs in the complex plane, each of which has the segment (a, b) on the real line as a chord. It states that, if P has at most one complex root in the region defined by the discs, then the sign-changes count `Bernstein_changes p a b P` ≤ 1.

Building on this theorem, we formalised the following termination proof. When the polynomial P input to `dsc` is square free, there is some minimum distance between the distinct complex roots of P. If we make $b - a$ sufficiently small, then at most one complex root fits into any of the three circles of the Three Circles theorem, and the theorem guarantees that `Bernstein_changes p a b P` ≤ 1. Hence, as bisection will eventually produce recursive calls with sufficiently small $b-a$ for any original a, b input with $a < b$, we have the termination statement:

```
theorem dsc_terminates_squarefree:
  fixes P :: "real poly"
  assumes "P ≠ 0"  "degree P ≤ p"  "p ≠ 0"  "square_free P"
  shows "⋀a b. a < b ⟹ dsc_dom (p, a, b, P)"
```

Here, `dsc_dom` is a predicate on tuples `(p,a,b,P)` of `dsc`'s input parameters that characterizes the values where the function terminates. It is automatically generated by Isabelle from `dsc`'s definition, along with associated induction and simplification rules [19].

3.2 Soundness and Completeness

To verify the correctness, we need to first show that the algorithm is sound. In other words, every output element is correct: either a root `(m,m)` with `poly P m = 0`, or an isolating interval

`(u,v)` with exactly one root in (u, v). We package this as `dsc_pair_ok P (u,v)`:

```
definition dsc_pair_ok :: "real poly  ⇒ (real × real)  ⇒ bool" where
  "dsc_pair_ok P I ↔
     ((fst I = snd I ∧ poly P (fst I) = 0) ∨
      (fst I < snd I ∧ roots_in P (fst I) (snd I) = 1))"
```

and then prove:

```
theorem dsc_sound:
  assumes "dsc_dom (p, a, b, P)"  "degree P ≤ p"  "P ≠ 0"  "a < b"
  shows "∀I∈set (dsc p a b P). dsc_pair_ok P I"
```

Next, we verify that the algorithm is complete, i.e. it does not miss any root inside (a, b):

```
theorem dsc_complete:
  assumes "dsc_dom (p, a, b, P)"  "degree P ≤ p"  "P ≠ 0"
  shows "⋀x. poly P x = 0 ⟹ a < x ⟹ x < b ⟹
           (∃I∈set (dsc p a b P). fst I ≤ x ∧ x ≤ snd I)"
```

Both proofs are by induction on the recursive structure of `dsc`.

4 Newtonian Acceleration

Sagraloff's NewDsc algorithm [28] accelerates the classical bisection method by combining Descartes' rule with occasional Newton steps. Intuitively, when an interval $I = (a, b)$ contains a tight cluster of roots, bisection may require many subdivisions before the Descartes test drops to 0 or 1; NewDsc tries to "jump into" the cluster using a Newton-style step, and otherwise falls back to bisection.

4.1 Block Selection, Newton Windows, and Fallback

As in Dsc, each recursive call computes $\nu =$ `Bernstein_changes p a b P`. If $\nu = 0$ we discard the interval, and if $\nu = 1$ we output (a, b) as an isolating interval. For $\nu \geq 2$, NewDsc tries two additional moves before bisection:

- Block selection (`try_block`). With width $w = b - a$ and a parameter $N \geq 2$, consider the two boundary blocks $B_1 = (a, a + w/N)$ and $B_2 = (b - w/N, b)$. If one block also has variation count ν, then it already provably contains all roots of P in (a, b), so we keep that block and increase N quadratically, replacing it by N^2.
- Newton windows `try_newton`. If block selection fails, we take one Newton step from an endpoint, using ν as a "multiplicity proxy" to estimate the correction. The Newton step from endpoint a calculates $\lambda = a + \nu P(a)/P'(a)$ as an estimate of the location of a root or root cluster, and checks whether the variation count for an interval of width w/N around λ is also ν. If so, the interval is returned and recursed on. Similarly, a Newton step from endpoint b explores $\lambda = b - \nu P(b)/P'(b)$. On recursion, N is again increased to N^2.

The block selection step captures root clusters that lie very close to the endpoints of (a, b), where the Newton step would not find them [28]. If both attempts fail, we fall back to bisection (including the midpoint-root check as in Dsc) and update N more conservatively. In the formalisation, this fallback update sets N to $\max(4, \lfloor\sqrt{N}\rfloor)$: taking the square root undoes the previous aggressive squaring after a failed jump, while the maximum keeps the block parameter from becoming too small to be useful.

Our Isabelle formalisation follows this structure directly:

```
function (domintros) newdsc ::
```

```
  "nat ⇒ real ⇒ real ⇒ nat ⇒ real poly ⇒ (real × real) list"
where
  "newdsc p a b N P =
     (let v = Bernstein_changes p a b P in
      if v = 0 then []
      else if v = 1 then [(a, b)]
      else
        (case try_blocks p a b N P v of
           Some I ⇒ newdsc p (fst I) (snd I) (N * N) P
         | None ⇒
           (case try_newton p a b N P v of
             Some I ⇒ newdsc p (fst I) (snd I) (N * N) P
           | None ⇒
             (let m  = (a + b) / 2; N' = max 4 (nat ⌊sqrt (of_nat N)⌋);
                  mid = (if poly P m = 0 then [(m, m)] else [])
              in mid @ newdsc p a m N' P @ newdsc p m b N' P))))"
```

4.2 Completeness and Sign Variations in the de Casteljau Triangle

Termination and soundness proofs of `newdsc` are similar to those of `dsc`, while completeness is a genuinely new difficulty. In `dsc`, the $\nu \geq 2$ branch recurses on both halves, so a root can never be "thrown away". In `newdsc`, the block/Newton branches recurse on a single chosen window I and discard the rest. To justify this step, we prove the splitting lemma from [3]:

```
lemma Bernstein_changes_split:
  assumes "a < c"  "c < b"  "degree P ≤ p"
  shows "Bernstein_changes p a c P + Bernstein_changes p c b P ≤
    Bernstein_changes p a b P"
```

Suppose one of the block/Newton branches succeeds with a chosen window $I = (u, v)$ and preserves the global variation count. Applying `Bernstein_changes_split` first at u and then at v yields `Bernstein_changes p a u P` + `Bernstein_changes p u v P` + `Bernstein_changes p v b P` ≤ `Bernstein_changes p a b P`. By construction of the chosen window, `Bernstein_changes p u v P` = `Bernstein_changes p a b P`, hence `Bernstein_changes p a u P` + `Bernstein_changes p v b P` ≤ 0. Since Bernstein variation counts are nonnegative, both discarded gaps have variation count 0, and therefore contain no roots. Now let x be an arbitrary root of P in (a, b). The preceding argument rules out $x \in (a, u)$ and $x \in (v, b)$, while the boundary cases $x = u$ or $x = v$ are excluded using a strengthened lemma. Thus x lies strictly inside the chosen window, and the induction hypothesis on the recursive call yields an output interval covering x. The splitting lemma can be proven by inspecting a triangle generated by the de Casteljau subdivision algorithm for Bernstein coefficients (as in Chapter 10.2 of [3]). As illustrated in Figure 1, the algorithm constructs a triangular array of coefficients where each intermediate element is a convex combination of its parents: $b_j^{(i)} := \alpha b_j^{(i-1)} + \beta b_{j+1}^{(i-1)}$ with $\alpha, \beta > 0$ and $\alpha + \beta = 1$. The geometric intuition behind the splitting lemma relies on the observation that inserting a convex combination between two adjacent real numbers can never introduce additional sign changes.

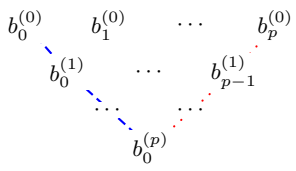

Figure 1 De Casteljau subdivision triangle. The top row contains the initial coefficients b, while the left boundary (blue, dashed) and right boundary (red, dotted) yield the resulting coefficients for the left and right sub-intervals, respectively.

It is worth noting that while there are no existing Isabelle formalisations of the de Casteljau algorithm, the algebraic correctness of the de Casteljau algorithm has been verified

in Rocq [4]. Their formalisation relies on point-wise recursive functions to compute sub-interval coefficients on the fly. While ideal for elegantly establishing algebraic equivalence, it appears to us that this functional formalisation is not well-suited for reasoning about the combinatorial properties of the coefficient lists, which is what we need. Therefore, we represent the two-dimensional triangle layer-by-layer as a list of lists, and prove that row transitions do not increase sign variations by strong induction on the coefficient list. The splitting lemma is then proven with another induction on the triangle.

5 Code Generation and Performance

Generating code directly from types like `real poly` or the underlying `rat poly` introduces a significant performance overhead, primarily stemming from the expensive greatest common divisor computations required for exact rational arithmetic. We decomposed the Descartes root-counting test into a sequence of operations, and adapted them to eliminate fractions.

5.1 Decomposing the Transformation

Recall that the coordinate transformation mapping the positive real line $\mathbb{R}_{>0}$ bijectively to an arbitrary open interval (a, b) is defined by the mapping $P_{(a,b)}(x) := (x+1)^n \cdot P\left(\frac{ax+b}{x+1}\right)$. In the theory file `Dsc_Taylor`, we formalise the observation that this monolithic transformation can be broken down into a sequence of four simpler operations [28]: (1) an initial Taylor shift by the lower bound a, yielding $P(x+a)$, (2) a linear coordinate scaling by the interval width $(b-a)$, (3) a reciprocal transformation via polynomial reversal, which replaces x with $1/x$ and multiplies by x^n to restore proper degrees, and (4) a final Taylor shift by 1. Our implementation works over coefficient lists and includes a list-based Ruffini algorithm for (1) and (4), and an iterative mapping loop for (2).

5.2 Eliminating Fractions

Instead of clearing fractions after each rational arithmetic step, we modified the components above by injecting scaling factors directly before each operation where a fraction would otherwise be introduced:

- Taylor shift by lower bound: When shifting the polynomial by the lower bound a, we consider its numerator and denominator ($a = n_a/d_a$). Rather than performing a rational shift, a function pre-scales each integer coefficient c_i by a corresponding power of the denominator (d_a^{k-i}, where k is the degree). This neutralizes the fractional denominators before they occur, allowing the subsequent shift to run exclusively as an integer operation.
- Scaling by interval width: Scaling the polynomial by the interval width $(b-a)$ introduces a new rational factor. Because the polynomial has already been scaled by d_a, the algorithm extracts the numerator and denominator of the adjusted width component directly: $(b-a) \cdot d_a = n_w/d_w$. The integer list is then multiplied by the numerator n_w. To clear the remaining denominator d_w, another pass of coefficient scaling is performed.

With these changes, all steps safely run without producing fractions, allowing us to verify an integer version of the Descartes' root test. It is then used in the construction and verification of `dsc_int` and `newdsc_int`, the partial tail-recursive integer versions of `dsc` and `newdsc`, suitable for code generation into SML.

5.3 Experimental Evaluation

For comparison against existing verified real root isolation algorithms, we extracted a Sturm-based function `isolate` developed by Joosten, Thiemann, and Yamada [17], which is the best performing verified real root isolation algorithm we are aware of in Isabelle. To ensure fairness, we customized their algorithm to remove extra steps in their original code, such as square-free factorization and packaging algebraic numbers.

For benchmarks, we reused 4 polynomials (P_1–P_4) randomly-generated in [23]. Moreover, we chose 4 Mignotte-type polynomials defined by $M_{n,\tau}(x) = x^n - 2(2^\tau x - 1)^2$. They are known for tight root clusters that make isolation very difficult. The tests were run in Poly/ML 5.9.2 on an Apple M4 CPU with 16 GB Memory. The results are shown in Table 1, with bold blue highlighting the fastest methods. The `dsc_exec` and `newdsc_exec` columns are for versions without the optimisations described in Sections 5.1 and 5.2. The numbers show these optimisations give a 10-200× speedup on our benchmarks.

Table 1 Wall-clock runtime (ms). Median of 5 runs or average of 100 runs (if runtime < 10ms).

	degree	isolate	dsc_int	newdsc_int	dsc_exec		isolate	dsc_int	newdsc_int	newdsc_exec
P_1	29	24200	**1.7**	8.7	34	$M_{16,32}$	3171	538	**56**	2195
P_2	29	826	**0.5**	1.5	6.7	$M_{32,16}$	14676	2311	**174**	15690
P_3	44	1115012	**7.2**	28	301	$M_{32,32}$	63228	10248	**449**	88617
P_4	44	3059	**0.8**	1.9	10	$M_{8,128}$	2500	493	**39**	731

The Sturm-based isolation algorithm struggled heavily because coefficient sizes explode during polynomial remainder operations. Our algorithms beat `isolate` on these randomly generated polynomials by several orders of magnitude. For Mignotte polynomials, the extremely tight clustering of roots forces a pure bisection approach like `dsc_int` to undergo an immense number of tree subdivisions. By generating occasional Newton steps to jump directly into the center of tight root clusters, `newdsc_int` collapses the depth of the subdivision tree, outperforming `dsc_int` on all Mignotte benchmarks. The extra steps introduce an overhead on simpler tasks (P_1–P_4) compared to pure bisection.

6 Conclusion and Future Work

We have fully verified a classical and a Newton-accelerated Descartes-based algorithm for real root isolation, proving their termination, soundness, and completeness. Brief experiments show promising performance. In future work, we wish to explore further optimisations such making use of an approximate model of computation [18]. In the end, we want to explore the possibility to refine the optimised algorithm into efficient LLVM code [20], achieving performance comparable to that of an unverified state-of-the-art algorithm.

7 Large Language Model Usage

LLMs helped us reduce a small amount of manual verification effort. We tried ChatGPT 5.1 Thinking on chatgpt.com when producing proofs for `newdsc`, giving it the corresponding proofs for `dsc` for reference. In particular, the soundness proof it generated was in very good shape with only a few errors that could easily be fixed with Sledgehammer [27, 6]. Other proofs it generated were less valuable and overall not worth trying. The SML program that was used to run the brief experiments in Section 5 was written with the help of ChatGPT.

References

1 Z3 source: Real root isolation via Descartes' rule of signs. https://github.com/Z3Prover/z3/blob/43791ebf2abeb91fbe78656dd122c0f28b61098c/src/math/polynomial/upolynomial.cpp#L2506-L2509, 2026. Z3 4.17.0.0, commit 43791eb.

2 Victor Aladjev. Computer algebra system Maple: A new software library. In Peter M. A. Sloot, David Abramson, Alexander V. Bogdanov, Jack J. Dongarra, Albert Y. Zomaya, and Yuriy E. Gorbachev, editors, *Computational Science — ICCS 2003*, pages 711–717, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/3-540-44860-8_73.

3 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Algorithms and Computation in Mathematics, 10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2nd edition, 2006. doi:10.1007/3-540-33099-2.

4 Yves Bertot, Frédérique Guilhot, and Assia Mahboubi. A formal study of Bernstein coefficients and polynomials. *Mathematical Structures in Computer Science*, 21(4):731–761, 2011. doi:10.1017/S0960129511000090.

5 Dario A. Bini and Leonardo Robol. Solving secular and polynomial equations: A multiprecision algorithm. *Journal of Computational and Applied Mathematics*, 272:276–292, 2014. URL: https://www.sciencedirect.com/science/article/pii/S037704271300232X, doi:10.1016/j.cam.2013.04.037.

6 Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT Solvers. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE 2011)*, CADE'11, pages 116–130, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-22438-6_11.

7 George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition–preliminary report. *ACM SIGSAM Bulletin*, 8(3):80–90, aug 1974. doi:10.1145/1086837.1086852.

8 George E. Collins and Werner Krandick. An efficient algorithm for infallible polynomial complex root isolation. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '92)*, pages 189–194. ACM, 1992. doi:10.1145/143242.143286.

9 Jose Divasón, Sebastiaan Joosten, René Thiemann, and Akihisa Yamada. A formalization of the Berlekamp-Zassenhaus factorization algorithm. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 17–29, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3018610.3018617.

10 Manuel Eberl. A decision procedure for univariate real polynomials in Isabelle/HOL. In *Proceedings of the 4th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2015)*. ACM, 2015. doi:10.1145/2676724.2693166.

11 Manuel Eberl. Descartes' rule of signs. *Archive of Formal Proofs*, December 2015. https://isa-afp.org/entries/Descartes_Sign_Rule.html, Formal proof development. URL: https://www.isa-afp.org/entries/Descartes_Sign_Rule.shtml.

12 Manuel Eberl and Peter Lammich. Verifying an efficient algorithm for computing Bernoulli numbers. In Yannick Forster and Chantal Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2025.35, doi:10.4230/LIPIcs.ITP.2025.35.

13 Alperen Ergür, Josué Tonelli-Cueto, and Elias Tsigaridas. Beyond worst-case analysis for root isolation algorithms. In *Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation*, ISSAC '22, pages 139–148, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3476446.3535475.

14 Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In *Functional and Logic Programming - 10th International Symposium, FLOPS 2010, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 103–117, 2010. 10th International Symposium

on Functional and Logic Programming, FLOPS 2010 ; Conference date: 19-04-2010 Through 21-04-2010. doi:10.1007/978-3-642-12251-4_9.

15 Michael Hemmer, Elias P. Tsigaridas, Zafeirakis Zafeirakopoulos, Ioannis Z. Emiris, Menelaos I. Karavelas, and Bernard Mourrain. Experimental evaluation and cross-benchmarking of univariate real solvers. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation*, SNC '09, pages 45–54, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1577190.1577202.

16 J. R. Johnson. Algorithms for polynomial real root isolation. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 269–299, Vienna, 1998. Springer Vienna. doi:10.1007/978-3-7091-9459-1_13.

17 Sebastiaan Joosten, René Thiemann, and Akihisa Yamada. A verified implementation of algebraic numbers in Isabelle/HOL. *Journal of Automated Reasoning*, 64, 03 2020. doi:10.1007/s10817-018-09504-w.

18 Alexander Kobel, Fabrice Rouillier, and Michael Sagraloff. Computing real roots of real polynomials ... and now for real! In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 2016)*, pages 303–310. ACM, 2016. doi:10.1145/2930889.2930937.

19 Alexander Krauss. *Defining Recursive Functions in Isabelle/HOL*. Technische Universität München, 2012. Part of the official Isabelle documentation distribution.

20 Peter Lammich. Generating verified LLVM from Isabelle/HOL. In *ITP 2019: Interactive Theorem Proving*, jun 2019. Interactive Theorem Proving, ITP 2019 ; Conference date: 08-09-2019 Through 13-09-2019. URL: https://itp19.cecs.pdx.edu/, doi:10.4230/LIPIcs.ITP.2019.22.

21 Wenda Li. The Budan-Fourier theorem and counting real roots with multiplicity. *Archive of Formal Proofs*, September 2018. https://isa-afp.org/entries/Budan_Fourier.html, Formal proof development. URL: https://www.isa-afp.org/entries/Budan_Fourier.html.

22 Wenda Li, Grant Olney Passmore, and Lawrence C Paulson. Deciding univariate polynomial problems using untrusted certificates in Isabelle/HOL. *Journal of Automated Reasoning*, 62(1):69–91, 2019. doi:10.1007/s10817-017-9424-6.

23 Wenda Li and Lawrence C. Paulson. Counting polynomial roots in Isabelle/HOL: A formal proof of the Budan-Fourier theorem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '19)*, pages 52–64. ACM, 2019. doi:10.1145/3293880.3294092.

24 Anthony Narkawicz, César Muñoz, and Aaron Dutle. Formally-verified decision procedures for univariate polynomial computation based on Sturm's and Tarski's theorems. *Journal of Automated Reasoning*, 54(4):285–326, 2015. doi:10.1007/s10817-015-9320-x.

25 Anthony Joseph Narkawicz, Cesar Munoz, and Aaron M. Dutle. A decision procedure for univariate polynomial systems based on root counting and interval subdivision. *Journal of Formalized Reasoning*, 11(1):19–41, Jan. 2018. URL: https://jfr.unibo.it/article/view/8212, doi:10.6092/issn.1972-5787/8212.

26 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. doi:10.1007/3-540-45949-9.

27 Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL 2010. The 8th International Workshop on the Implementation of Logics*, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2012. URL: /publications/paper/wV, doi:10.29007/36dt.

28 Michael Sagraloff. When Newton meets Descartes: A simple and fast algorithm to isolate the real roots of a polynomial. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 2012)*, pages 297–304. ACM, 2012. doi:10.1145/2442829.2442872.

29 The Mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM*

SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), pages 367–381. Association for Computing Machinery, 2020. doi:10.1145/3372885.3373824.

30 The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.8)*, 2025. https://www.sagemath.org.

31 René Thiemann and Akihisa Yamada. Polynomial factorization. *Archive of Formal Proofs*, January 2016. https://isa-afp.org/entries/Polynomial_Factorization.html, Formal proof development. URL: https://www.isa-afp.org/entries/Polynomial_Factorization.shtml.

32 Fox Thomson and Wenda Li. The theorem of three circles. *Archive of Formal Proofs*, August 2021. https://isa-afp.org/entries/Three_Circles.html, Formal proof development. URL: https://www.isa-afp.org/entries/Three_Circles.html.

33 Freek Wiedijk. Formalizing 100 theorems. https://www.cs.ru.nl/~freek/100/, 2024. Accessed: February 14, 2026.

34 Julianna Zsidó. Theorem of three circles in Coq. *Journal of Automated Reasoning*, dec 2013. 25 pages, 5 figures. URL: https://inria.hal.science/hal-00864827, doi:10.1007/s10817-013-9299-0.