Computer-assisted proofs

Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Nordbergstr. 15, A-1090 Wien, Austria
Arnold.Neumaier@univie.ac.at
http://www.mat.univie.ac.at/∼neum/

Abstract

This paper discusses the problem what makes a computer-assisted proof trustworthy, the quest for an algorithmic support system for computer-assisted proof, relations to global optimization, an analysis of some recent proofs, and some current challenges which appear to be amenable to a computer-assisted treatment.

1 Introduction

Computer-assisted proofs have a long history. Perhaps one of the oldest computer-assisted proofs dates back to the year 1610:

Theorem (Ludolf van Ceulen 1610)
3.14159 26535 89793 23846 26433 83279 50288 is smaller than π by less than 10^{-35}.

Proof. Apply the algorithm of Archimedes until the error is small enough. See `http://mathblogger.free.fr/index.php?m=01&y=08&entry=entry080128-103425` □

(Laurent Thery noticed that the last 5 digits 50288 were misprinted in the published version of the present paper by a copying mistake, resulting in a second group of 26433.)

At that time, computers were human, slow and not very reliable. The running time on the biological computer quoted as the author of the theorem was over 14 years; the amount of external storage (paper) used is not recorded.

The algorithm of Archimedes was well-known since antiquity. But nobody checked the correctness of van Ceulen's implementation of it, nor inspected anyone the logfiles produced (probably they were not even kept as record). Nevertheless, the theorem is correct and was always regarded as correct.

Improved error bounds, using a much faster algorithm, were computed (on a similar computer) by SHANKS 1893. Shanks approximated π to 707 decimal digits. Again, nobody checked the correctness of the proof or the implementation. However, this time, although based on an algorithm known to be correct, the theorem turned out – much later – to be false (FERGUSON 1945): Only the first 527 digits (and a few random later digits) were correct. It didn't matter – nobody needed (or needs today) so many digits of π. But it is a scientific challenge to get things 100% right – even things whose only use is to satisfy the human curiosity. Today, over 10^9 digits of π are believed to be known. (See `http://zenwerx.com/pi.php` .)

Trusting computations. All our communication, and all our knowledge, is based on trust. Trust in the sources, trust in the tools used to create and/or check statements, trust in one's memory and in one's reasoning power. And trust in computations, if these play a role.

Trust is important in all proofs, not only the computer-assisted ones. We trust the referees, the experts in the fields, the quoted references, or our own expertise. Sometimes, the trust turns out to be unjustified later. Nevertheless, without trust there is no knowledge, for everything can be doubted. In particular, we trust computations

- if the algorithms on which they are based are understandable and have understandable correctness proofs,
- and if they are carried out with a degree of care that – based on a finite amount of past experience on similar computations – we learnt to rate as reliable.

In case of doubt, we may want to have independent implementations with which the computations can be done. Preferably these implementations should be based upon different algorithms to increase our confidence in the correctness.

The Pentium bug a few years ago shows that computer hardware cannot be trusted unconditionally. Nevertheless, we expect that significant bugs are sooner or later found and then corrected. The search for bugs usually begins with

getting an unexpected contradiction to supposed behavior, followed by debugging – the search for the piece in the long chain of arguments or instructions which had no true justification. After the corrections everything depending on the correction must be repeated. For mistakes in crucial places (such as in Shank's calculation of π) on which many other things depend, a large part of the results may be wrong.

On the other hand, in robust, well-structured mathematical proofs, there are many connections between the concepts and arguments used. This has the consequence that most results remain correct when a particular link is found wanting. For example, Russell's paradox invalidated the logical basis of set theory, but hardly affected the bulk of mathematics. This explains that mathematics as a whole thrives, in spite of many wrong published theorems. But it shows the importance of plausibility checks and/or formal verification tools.

Proof checking. Checking a proof ist often (not always) much easier than finding it. Finding the proof of the following result took at the time about 150 days of (human) computer time. Checking it was a matter of 20 minutes.

Theorem. (Cole 1903)
$2^{67} - 1$ is not a prime.

Proof. $2^{67} - 1 = 761838257287 * 193707721.$ □

Most mathematicians regard this as a perfectly valid proof, although the details are long and tedious, and error prone (but doable) for manual checking. But the underlying algorithm is familiar to many people (i.e., for the present purposes, biological computers with independent implementations of the same algorithm) so that trust is granted easily.

The traditional check of numerical calculations with a pocket calculator happens to give the same results for the left and right side of the equation in the proof. We know, however, that pocket calculators are not reliable, and that rational multiprecision arithmetic or interval arithmetic are needed to guard against arithmetical errors in computer calculations.

No one ever will probably want to check the details of a proof of the following theorem, proved (using the formal proof system Coq) by long computations:

Theorem. (CAPROTTI & OOSTDIJK [1])
The 40 decimal digit number

90262 58083 38499 68604 49366 07214 23078 01963
is a prime.

Thus having reduced a mathematical problem to algorithmic computations together with a credible execution of the computations is generally considered enough evidence to count as a proof – except when the algorithms employed are unfamiliar.

When the four-color-theorem was first proved in 1976, the heavy computational part (weeks of combinatorial computations) created strong sentiments about whether it was a valid proof. In the mean time, reservations largely subsided. The computational part of the four-color-problem has become a test problem in mixed-integer linear programming, (with 1372 binary variables and 4944 linear inequality constraints). See FERRIS et al. [3]. Although still nontrivial to solve, its mathematical complexity is now embedded in a well-tested algorithmic framework with many independently written computer codes. Thus it is tamed, understood, even though the details are executed by a computer.

The key for routine acceptability is therefore to reduce a problem to one which can be settled by computer using general purpose software available to many and used by many.

2 The structure of computer-assisted proofs

Typically, computer-assisted proofs which don't consist of pure computation proceed in three stages, which are not always clearly separated:

Stage 1. Qualitative conceptual understanding reduces the problem to a finite number of individual subproblems, each characterized by a small number of parameters.

Stage 2. The analysis of each subproblem results in a number of equations and/or inequalities satisfied for a counterexample.

Stage 3. A specialized computational algorithm is executed, showing that the resulting constraint satisfaction problems have no solution.

The third stage is completely independent of the remainder of the proof. But the formulation of the constraints that make the third stage computationally tractable may involve lots of trial and error.

This leads to the challenge to create software which handles the third step stage automatically and reliably, without the need on the user's side to understand the details. This will allow nonexpert users to pose the computational part of a desired proof in a straightforward (close to) mathematical language, and then run the verification software to check whether the wanted statement can in fact be algorithmically verified.

Such a software system would relieve users from having to think about how to do the computations, so that they can concentrate on the formulation of the constraints. If the software is fast enough, it would allow users to try many formulations and select the the most successful one.

In addition, this would allow computer-assisted proofs to become much more transparent than at present (where the computational and the conceptual parts of the proofs are typically intermingled). It would also make independent checks much easier since the statements which are to

be checked by computation are cleanly separated from the conceptual arguments.

Computer-assisted proofs in analysis always require interval methods; see, e.g., [4, 10, 12]. But the typical user of such a verification system should need to know as little about the intricacies of reducing overestimation in interval computations as the typical player of a computer game knows about the underlying computational geometry.

The success of Matlab as an algorithmic language for approximate numerical calculations is largely due to the simplicity with which mathematical algorithms can be encoded. The success of LaTeX as a mathematical typesetting language is largely due to the flexibility with which traditional mathematical notation can be encoded. Therefore, the part of the verification software which the user sees should be as simple to use as a Matlab editor and flexible enough that not much more than elementary LaTeX knowledge is needed to use the software.

The double bubble proof. To see what such a system must be able to solve, I made a study of the computer-assisted proof of the double-bubble conjecture by HASS et al. [6]. From the perspective of the interval community, a short synopsis of the problem and its solution was given by Andreas FROMMER [4]. It is not easy to separate the computationally proved part into a mathematical statement involving only the conditions to be checked without further reference to the geometric sources. Such a separation is needed, however, to enable experts in computer verification but not in geometry to make an independent check of the computational part of the proof.

Here is how I understand what they proved by computer, and what together with the conceptual arguments given in their paper suffices to establish the validity of the double bubble conjecture. (Their notation translates to my notation as $y_1 = s_1, y_2 = s_2$.)

Theorem.
The following conditions are inconsistent:

$$c_1^2 + s_1^2 = c_2^2 + s_2^2 = 1,$$

$$s_1 \geq 0,\ \ s_2 \geq 0,\ \ c_1 \geq 0,\ \ |c_2| \leq 0.5,$$

$$-f_i = f_0 = us_1^2 + s_1c_1\sqrt{3} = us_2^2 + s_2c_2\sqrt{3},$$

$$h_0 = 1 + u,\ \ h_i = 1 - u,\ \ u \in [-1, 9],$$

$$\int_{y_1}^{y_2} \frac{t_0(y)}{\sqrt{4y^2 - t_0(y)^2}}|dy| = \int_{y_1}^{y_2} \frac{t_i(y)}{\sqrt{4y^2 - t_i(y)^2}}|dy|$$

$$\int_{y_1}^{y_2} \frac{y^2t_0(y)}{\sqrt{4y^2 - t_0(y)^2}}|dy| = \int_{y_1}^{y_2} \frac{y^2t_i(y)}{\sqrt{4y^2 - t_i(y)^2}}|dy|$$

where

$$t_0(y) = h_0y^2 - f_0,\ \ t_i(y) = h_iy^2 - f_i.$$

Here the absolute values indicate that the integral is to be understood as a line integral along a path from y_1 to y_2 that reverts its direction at points where the square root vanishes. (It is not difficult to see that this implies that the singularity is harmless.)

Clearly, this theorem needs no geometric knowledge for its proof. It is also nearly obvious (at least for those knowledgeable in interval techniques) that this theorem, if true, can be proved in a finite computation using interval techniques, by implementing a branch and bound process that computes enclosures for both sides of each equation, and discards a box if the resulting intervals are disjoint.

Assuming the theorem to be correct, the success of the method is guaranteed if the enclosures have overestimations that tend to zero with the box size, and this is easy to achieve. The actual implementation of Hass and Schlafly indeed uses only the simplest tools (Riemann sums, interval evaluations, and a limited form of constraint propagation), and finishes the work after having considered 15016 boxes. (More sophisticated strategies would probably shorten the computer-assisted part of the proof considerably.)

Note that formulating the theorem (and adding the above comments) makes the whole proof much more transparent than the proof as actually given. Every reader can see what is the geometric and what is the algebraic part of the proof. Thus, geometers could content themselves checking the geometric part and just trust the programs to settle the algebraic part; the numerical analysts could trust the geometry and check the algebra. In this way the checking work is naturally divided, and the proof easier to check. This adds to the credibility and thus to its trustworthiness. At present one needs to know both sides of the expertise to be able to check the proof.

Had a reliable software system been widely available that allows to check proposed theorems of the above form, the authors would probably have used it – or would have been encouraged by the referees to use it –, to the benefit of all. The availability of such a verification system would make people elsewhere aware of the possibilities of automatic verification, and would give easy access to powerful techniques which are currently for many potential users still a big hurdle.

3 Relations to global optimization

In practice, one often wants to assert something specific about concrete problems in analysis (as opposed to general theorems about all objects of a certain type) To do this rigorously requires the ability to reason with constructively given elements or sets in $\mathbb{R}$, $\mathbb{R}^n$, or some function space. These are typically defined in terms of inequalities (e.g., balls as sets of points x with $\|x - x_0\| \leq r$). Thus we need to be able to check whether certain equations and/or inequalities

imply others. Thus we need the ability to work efficiently with equations and inequalities. The assertion

$$A(x) <= B(x) \quad \Rightarrow \quad f(x) < g(x)$$

for vector-valued functions A, B with componentwise inequality and real-valued functions f, g is equivalent with the assertion that the system of inequalities

$$\begin{pmatrix} B(x) - A(x) \\ f(x) - g(x) \end{pmatrix} \geq 0$$

has no solution.

Thus we need to be able to verify rigorously the nonexistence of solutions of systems of inequalities. *Traditional numerical software systems cannot do this – not even approximately – in the absence of special properties such as linearity or convexity.* (Traditional nonlinear programming software just spends some time trying to find a solution and then gives up without a conclusion.) Instead, one needs techniques based on interval arithmetic [10].

The problems stated are semilocal or even global in nature: They must be solved with quantifying over sets with substantial volume (the semilocal case) or even over unbounded sets (the global case).

Finding *best* inequalities leads to global optimization problems. For polynomial optimization problems, proofs of global optimality, and hence of the validity of inequalities, are possible due to necessary and sufficient global optimality conditions recently derived by NEUMAIER & SCHICHL [14]. The proof is based on a deep result from real algebraic geometry, the so-called Positivstellensatz. This, in turn, can be proved using nontrivial techniques from model theory.

The conditions lead to a certificate of optimality, consisting in the coefficients of a polynomial equation. The difficulty of proof finding is reflected in the potentially exponential size of the equation; but in practice often a small certificate exists.

GloptLab [2] is a Matlab implementation of a branch-and-bound method currently under development at the University of Vienna, mainly being implemented by Ferenc Domes. It currently solves quadratic constraint satisfaction problems defined by bound constraints and linear and quadratic equations and inequalities. Every algebraic system of equations and inequalities can be brought into this form, but integrals (such as those needed in the double bubble proof) are not covered at present. GloptLab is based on constraint propagation combined with the maximal exploitation of quadratic (hence small) certificates of infeasibility related to the above results.

Theorem.
For every positive semidefinite matrix G and every matrix $Z \leq 0$, the polynomial $p(x)$ defined by

$$\begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} = \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix} + z^T Q(x) + p(x)$$

satisfies $0 \leq p(x)$ on the feasible set.

In place of longer polynomial certificates, case distinctions (branching steps) are used. Compared with other solvers for global optimization and constraint satisfaction, the emphasis is on mathematical rigor and short proofs, i.e., a small number of branching steps. The implementation uses the Intlab package (RUMP [11]) for interval calculations, the semidefinite programming package SeDuMi [15] for finding certificates, and converters from the COCONUT environment (SCHICHL [13]) to solve global optimization problems posed in the modeling language AMPL.

Rounding error control is essential to make the computations mathematically rigorous. For example, a strictly convex, quadratic inequality constraint defines an ellipsoid whose interval hull is easy to compute analytically. But to cope efficiently with rounding errors is nontrivial. GloptLab uses an algorithm which computes a directed Cholesky factor for a symmetric, positive definite interval matrix $\mathbf{A}$, i.e., a nonsingular triangular matrix R such that $A - R^T R$ is positive semidefinite for all $A \in \mathbf{A}$, and tiny if the components of $\mathbf{A}$ are narrow intervals. In particular, $x^T A x \geq \|Rx\|_2^2$, with little overestimation. This allows us to write strictly convex, quadratic inequalities in the form $||Rx||_2^2 - 2a^T x \leq \alpha$. By solving $R^T R x_0 = a$, we find bounds $\|R(x - x_0)\|_2 \leq \delta := \sqrt{\max\{0, \alpha + a^T x_0\}}$, from which one can find the componentwise bounds $|x - x_0| \leq \delta \operatorname{diag}(R^{-1} R^{-T})$.

4 Open conjectures

We end by listing some open conjectures which are likely to be tractable in the near future by computer-assisted techniques.

The Wright conjecture (1955).
For $0 < p < \pi/2$, every solution of the delay-differential equation $x'(t) = -p(1 + x(t))x(t - 1)$ tends to zero as $t \to \infty$.

This conjecture by WRIGHT [17] is currently under attack in Szeged, and appears to be "almost" (i.e., up to final correctness checks) solved.

Hilbert's 16th Problem, second part (1900).
"This is the question as to the maximum number and position of Poincaré's boundary cycles (cycles limites) for a differential equation of the first order and degree of the form

$$\frac{dy}{dx} = \frac{Y}{X},$$

where X and Y are rational integral functions of the n-th degree in x and y. Written homogeneously, this is

$$X(y\frac{dz}{dt} - z\frac{dy}{dt}) + Y(z\frac{dx}{dt} - x\frac{dz}{dt}) + Z(x\frac{dy}{dt} - y\frac{dx}{dt}) = 0,$$

where X, Y, and Z are rational integral homogeneous functions of the n-th degree in x, y, z, and the latter are to be determined as functions of the parameter t." (quoted from HILBERT [7]). The problem has been reposed in 2000 by SMALE [16] as one of the "Mathematical Problems for the Next Century". For a recent survey, see ILYASHENKO [8].

It is known that the number of limit cycles is finite in each particular instance, but no finite upper bound is known. This problem is still unsolved, even for $n = 2$, in spite of a lot of research on the problem. For $n = 2$, the largest number of limit cycles known is 4, and probably this bound is tight. For $n = 2$, the problem amounts to understanding the set of periodic solutions of pairs of autonomous differential equations

$$\dot{x} = ax^2 + bxy + cy^2 + dx + ey + f,$$

$$\dot{y} = a'x^2 + b'xy + c'y^2 + d'x + e'y + f'.$$

This is a 12-dimensional manifold of orbits, which can be reduced to 5 dimensions by suitable linear transformation of the variables. Periodic solutions can be described as fixed points of Poincaré maps. Hence the problem is to show that certain equations $F(z, w) = z$ in the single variable z and parameterized by a 5-dimensional vector w never have more than 4 solutions (except when they have infinitely many – a degenerate, well understood case excloded by the demand that the periodic solution is a limit cycle).

The low dimension makes the problem appear feasible for branch and bound techniques. Some obstacles remain: The fixed point equation is not algebraic but a Poincaré mapping must be enclosed, and the parameter vector ranges over an unbounded 5-dimensional manifold, whence additional asymptotic estimates are needed.

The Thompson-Smith problem (2000).
Is 10 or 12 the minimim norm of the Thompson-Smith lattice in dimension 248?

The conjectured answer 12 (for background information see LEMPKEN et al. [9]) would follow (according to Gabriele Nebe, personal communication) from the solvability of a nonlinear mixed integer constraint satisfaction problem. The integer coefficients are generated automatically by Maple; some have several hundred digits.

5 Conclusion

As we have seen, there is lots of interesting work to be done at the interface between

- **Mathematics** – the art and science of unambiguous concepts,
- **Logic** – the art and science of impeccable reasoning,
- **Operations Research** – the art and science of optimal planning,
- **Numerical Analysis** – the art and science of algorithmic approximation, and
- **Computer Science** – the art and science of efficient computation.

References

[1] O. Caprotti and M. Oostdijk, Formal and Efficient Primality Proofs by Use of Computer Algebra Oracles, J. Symbolic Computation 32 (2001), 55-70.

[2] F. Domes and A. Neumaier, GLOPTLAB – a Matlab-based global optimization laboratory. In preparation (2007).

[3] M.C. Ferris, G. Pataki and S. Schmieta, Solving the Seymour problem, Optima 6 (2001), 2.

[4] A. Frommer, Proving conjectures by use of interval arithmetic, pp. 1–13 in: Perspective on Enclosure Methods (U. Kulisch et al., eds.), Springer, Wien 2001.

[5] T.C. Hales, A Proof of the Kepler Conjecture, Annals of Mathematics 162 (2005), 1065-1185.
Computations are described in the special issue: Discrete and Computational Geometry, 36 (2006), 1-265.

[6] J. Hass, M. Hutchings, and R. Schlafly, Double bubbles minimize, Ann. Math. (2) 515 (2000), 459–515.
`http://math.ucdavis.edu/~hass/bubbles.html`

[7] D. Hilbert, Mathematical Problems, Bull. Amer. Math. Soc. 8 (1901/2), 437–479.

[8] Yu. Ilyashenko, Centennial history of Hilbert's 16th problem, Bull. Amer. Math. Soc. 39 (2002), 301-354.

[9] W. Lempken, B. Schröder and P.H. Tiep, Symmetric Squares, Spherical Designs, and Lattice Minima, J. Algebra, 2001

[10] A. Neumaier, Interval methods for systems of equations, Cambridge Univ. Press 1990.

[11] S.M. Rump, INTLAB – INTerval LABoratory, pp. 77–104 in: Developments in reliable computing (T. Csendes, ed.), Kluwer, Dordrecht 1999.
http://www.ti3.tu-harburg.de/rump/intlab/index.html

[12] S.M. Rump, Verification methods for dense and sparse systems of equations, pp. 63-136 in: J. Herzberger (ed.), Topics in Validated Computations – Studies in Computational Mathematics, Elsevier, Amsterdam 1994.

[13] H. Schichl, The COCONUT environment.
http://www.mat.univie.ac.at/coconut-environment/

[14] H. Schichl and A. Neumaier, Transposition theorems and qualification-free optimality conditions, SIAM J. Optimization, to appear (2006).
http://www.mat.univie.ac.at/∼neum/papers.html#trans

[15] SEDUMI web site.
http://sedumi.mcmaster.ca/

[16] S. Smale, Mathematical Problems for the Next Century, Math. Intelligencer 20, No. 2, 7-15, 1998.

[17] E. M. Wright, A nonlinear difference-differential equation, J. Reine Angew. Math. 194 (1955), 66-87.