Computers: (Ac)counting for Mathematical Proofs

A.M. Cohen

1. Introduction

Some forty-five years ago, it became apparent that one could count on computers for computing. Many engineers, physicists and others, but few mathematicians, seemed to be aware of it.

For instance, numerical analysis arose from the need outside of mathematics to perform elaborate computations. The finite element method has been conceived by engineers (cf. e.g. [10]). Many numerical results dating from before 1950 are due to physicists. Not until forty years ago, numerical analysis came alive as a part of mathematics.

History repeated itself about thirty years ago, when it became clear to physicists and computer scientists that exact (symbolic) computations could be carried out successfully on a computer. Again, it took a few years before software and computer power had been developed to the point where even mathematicians could be impressed to the extent that they wanted to use it. In 1985 that point was reached, and subsequently computer algebra was being incorporated into mathematics. Over the last decade, computer algebra has grown into a part of mathematics.

A pattern emerges, according to which computers are being used in disciplines outside of mathematics, in a way that mathematicians will learn to appreciate only several years later, when others have shown its success.

A third to follow in line with this pattern might be automatic proof verification. Except for a single, although quite significant, exception like N.G.

de Bruijn, formal proof checking was not taken seriously by mathematicians over the last twenty years. In the meantime, logicians and computer scientists have developed software tools for verifying formal proofs quite thoroughly. So much so that the use is becoming of some interest to mathematicians.

Extrapolating from the experiences with numerical analysis and computer algebra, it seems only natural to predict that, if the imminent success of these automatic proof verifiers persists, mathematicians will also count this field as a part of mathematics.

Each of these three disciplines, numerical analysis, computer algebra and proof verification, comes with field specific software. In this article I want to highlight just one aspect of this software, namely the question how the classical notion of proof should be interpreted now that, due to computers, new methods of computation and verifying proofs become available.

1.1. Numerical mathematics

The last few years we have seen an increase of interactions between computer algebra systems and numerical software. Usually the numerical software is used as a library in a computer algebra environment. In the context of proofs it is of considerable interest to explore how numerical mathematics can be used to compose both efficient and exact computations.

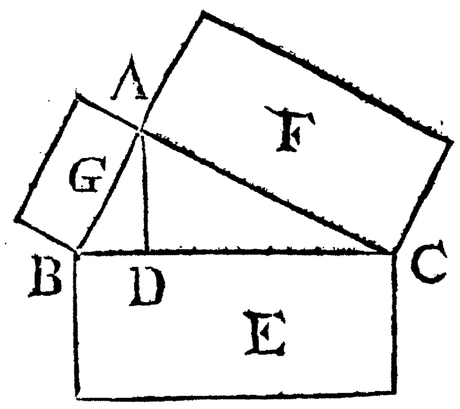
For example, finding the sign of a certain real algebraic number can be done by purely algebraic methods which are far more inferior in time performance than numerical methods. The optimal strategy seems to use exact arithmetic to determine the precision necessary for computations with an approximation of the algebraic number to result in the same sign as the exact computation. Numerical software with the required precision will then finish the job much more efficiently than exact arithmetic can.

Because numerical software is not made for exact computations, but also because of my ignorance regarding the topic, I will leave out numerical mathematics from most of the discussion and concentrate on proofs in computer algebra and verification.

2. TWO KINDS OF PROOFS

As opposed to the early times of computers, when their use demanded a thorough knowledge of commands, idiom, a plethora of patience and perseverance, computers are now pleasant, playful, instructive, and sometimes even efficient tools for mathematicians. The computer lends itself to all kinds of mathematical experiments, computations and visualizations, which can lead to interesting conjectures and experimental circumstantial evidence. But, next to striking claims, our strong mathematical tradition also demands proofs, and my interest is in finding out what help the computer can offer in this direction.

PROP. XXXI. THEOR.



In rectangulis triangulis BAC *figura* E, *quae fit a latere* BC, *rectum angulum* A *subtendente, aequalis est eis* F + G, *quae a lateribus rectum angulum comprehendentibus fiunt, similibus & similiter descriptis.*

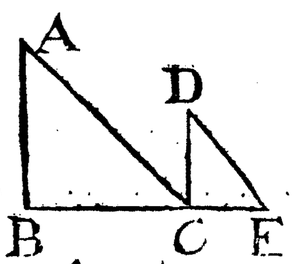
α. cor. 8. 6. β. 2. cor. 20. 6. Duc perpendicularem AD: & erit α ∴ BC, BA, BD, item ∴ BC, CA, DC. Hinc β BC: BD = E: G, & BC: DC = E: F, vel inuerse DC: BC = F: E, & BD: BC = G: E. Er-

γ. 24. 5. δ. sch. 14. 5. go γ BD + DC: BC = F + G: E. Sed BD + DC = BC: ergo δ F + G = E. Q.E.D.

Aliter.

ε. 1. cor. 20. 6. ζ. 47. 1. F: E = ε (CA: BC)² = ε CAq: BCq, & G: E = (AB: BC)² = ABq: BCq. Ergo γ F + G: E = CAq + ABq: BCq. Sed CAq + ABq = ζ BCq. Ergo F + G = δ E.

PROP. XXXI. THEOR.



Si duo triangula ABC, DCE, *quae duo latera duobus lateribus proportionalia habent* (BA: AC = CD: DE), *componantur secundum unum angulum ita, vt homologa latera ipsorum* BA & CD, item AC & DE, *sint parallela: reliqua trian-*

Figure 1. Fragment of Euclid's Elements (around 300 B.C.), the classical book based on deductive proof with the axiomatic method (J.F. Gleditsch, Leipzig, 1743. Courtesy W.J.L. Nieland).

The classical notion of proof (see also figure 1) is that of a chain of steps, each of which is understandable to our colleagues, forming a deduction of a given assertion from axioms according to a certain logic. A necessary condition for the acceptance of an assertion as a theorem is the validation of a proof of it by a number of colleagues. My starting point is not so much to overthrow this classical notion as to examine how it should be interpreted in the context of new appearances of proofs due to the surge of powerful computers.

The most obvious appearance lies in arithmetic: a proof based on a certain amount of arithmetic, so enormous that it cannot be performed by hand, will be called a *computational proof* if it can be worked out by computer. But should any series of computations be accepted as proof? To this question I will devote considerable attention.

The second appearance is in the form of *formal proofs*, which are so elaborate and precise that they can actually be read and verified by a computer program (often called the *proof checker*, the procedure is called *automated verification*). Here the question I want to address is: of what use could such a system be for mathematics or mathematicians?

3. AUTOMATED VERIFICATION

I will start with formal proofs. Each step in such a proof follows from precisely indicated axioms by use of precisely indicated deduction rules. It is characteristic of the elaborate texts representing such a proof that they can be verified automatically, with existing software like LEGO.

3.1. Example

By way of illustration, I present a formal proof for the claim that 13 is a prime number.

definitions:

divides$(m : \mathbf{N}, n : \mathbf{N}) = \exists\, p : \mathbf{N}.\ p * m = n.$
prime$(p : \mathbf{N}) = \forall\, q : \mathbf{N}.\ \text{divides}(q, p) \rightarrow (q = 1) \vee (q = p).$
axiom1: $\forall\, i, j, k, l : \mathbf{N}.\ ((i = j * k + l) \wedge (0 < l < k)) \rightarrow \neg(\text{divides}(k, i)).$
axiom2: $\forall\, i : \mathbf{N}.\ (\forall\, j : \mathbf{N}.\ 1 < j \leq \text{Sqrt}(i)$
$\rightarrow \neg(\text{divides}(j, i))) \rightarrow \text{prime}(i).$
axiom3: $\forall\, i : \mathbf{N}.\ 1 < i \leq \text{Sqrt}(13) \rightarrow (i = 2) \vee (i = 3).$
axiom4: $\forall\, i : \mathbf{N}.\ (i = 2) \rightarrow (13 = 6 * i + 1) \wedge (0 < 1 < i).$
lemma5: $\forall\, i : \mathbf{N}.\ (i = 2) \rightarrow \neg(\text{divides}(i, 13))$. (axiom1, axiom4)
axiom6: $\forall\, i : \mathbf{N}.\ (i = 3) \rightarrow (13 = 4 * i + 1) \wedge (0 < 1 < i).$
lemma7: $\forall\, i : \mathbf{N}.\ (i = 3) \rightarrow \neg(\text{divides}(i, 13))$. (axiom1, axiom6)
lemma8: $\forall\, i : \mathbf{N}.\ 1 \leq i \leq \text{Sqrt}(13) \rightarrow \neg(\text{divides}(i, 13)).$
(axiom3, lemma5, lemma7)
conclusion: prime(13). (axiom2, lemma8)

In order to bound the size of this example, I have allowed for 1 to be a prime, and have made the following assumptions, which appear as axioms in the proof:

- If $i = j * k + l$ and $0 < l < k$, then k does not divide i (axiom1),
- It suffices for the proof of our claim to check that no natural number less than or equal to (the floor of) $\sqrt{13}$ and bigger than 1 divides 13 (axiom2),
- the only natural numbers less than or equal to (the floor of) $\sqrt{13}$ and bigger than 1 are 2 and 3 (axiom3),
- $13 = 6 * 2 + 1$ (axiom4), and
- $13 = 4 * 3 + 1$ (axiom6).

In order to increase human readability, I have not given the literal input into the proof checker LEGO, but a palatable version that still gives the flavour. After reading the text, LEGO will return a 'check mark', saying that the proof is accepted. If some step in the proof has not been derived using the specified deduction rules, the program stops and reports where it got stuck.

This example might give the impression that formal proofs consist of unwanted compilations of trivia. But, regardless how big the bulk of formalities may seem in the above example, in general it is expected to be a linear function of the length of the classical proof. Thus, when the first classical proofs can be dealt with in formal guise, much bigger ones will follow suit.

Also, there are clear indications that formal proof checkers, just like people, can work with meta-theorems, which will cater for considerable size reductions of the formal proof. (Compare with the use of macros in source texts of programs like TEX.)

In short, due to such favourable developments proof checkers may turn into serious candidates for everyday mathematical use.

3.2. The use of proof checkers

But, you may wonder, what then is the use mathematicians can make of these automated verifiers? I will list four ways in which I envisage a role for the proof checker.

1. There are indications of the practical use that automated verification may have. In computer science for instance, simple communication protocols which are being used have been proved correct (cf. [4, 8, 9]). Especially for hardware like processor chips, which are produced in large quantities, it can be economically justified to devote plenty of time, energy and money to

increase the reliability of the procedures that are being baked into the chip. Automated verification may be laborious, calling the whole production back to the factory is likely to cost excessively more. If the well-publicized mistake with the Pentium chip was indeed due to an error in the mathematical design for division, an automated verifier might have detected it before production. On the other hand, if, as other rumours have it, the mistake was due to some pieces of data falling off the blueprint when being copied, no automated verifier would have helped.

2. The mere fact that it is possible to verify a proof automatically, brings about a challenge to actually supply such a proof. If proof verification were to enable mathematicians, in exchange for somewhat greater precision and elaboration, to formalize their work in a new generation of 'Bourbakism', then this might well become the standard. Let us not forget that today's proofs are much more rigorous than those of previous eras.

3. The formal proof verifier can be of service when putting together or restructuring a complicated proof. By using the proof checker interactively, and declaring axioms all intermediate steps that haven't been proved yet and are deemed necessary (in much the same way we did in the proof that 13 is prime), we can dynamically develop a strategy for finding the proof of the full claim.

4. This interactive use can also be of significance to education. In a computer learning environment, the proof checker might offer the necessary structure for helping the student to realize a proof and for verifying intermediate results.

Presently, good human interfaces are lacking. Before the mathematician will successfully employ formal proof checkers, the link with daily mathematical use should be much more direct. One of the necessary ingredients for achieving this is 'the mathematical vernacular', as suggested by De Bruijn: a language so formal that it can be used as input for a proof checker, yet so close to everyday language that it produces human readable texts.

4. THE COMPUTATIONAL PROOF

The characteristic property of a *computational proof* is that a lot of arithmetic is involved of the kind that is easy to automate. On the one hand, invoking such a proof implicitly acknowledges the lack of a better one: the mathematician's sense of beauty favours a short proof with little arithmetic involved over a computational proof.

On the other hand, it does not imply that the proof is necessarily a dull chain of calculations. For instance, there are much more subtle computational proofs of the claim 'n is prime' for a specified natural number n than the most obvious one, which is based on the equivalence of primality of n with the truth of the assertion:

for each natural number k between 1 and $\sqrt{n}$ we have $\gcd(n, k) = 1$.

We then let the computer determine all possible values for k and verify, for each of those values of k, that $\gcd(k, n) = 1$. If the prime number n is a 31 digit number, then this verification entails more than 10^{15} gcd computations, amounting to more than 10^8 years of computation. But present day software provides a verification of this fact in a few seconds.It is an art to develop such fast computational proofs; experience has shown that, as a byproduct of this activity, surprising mathematical theorems may emerge.

I have already asserted that a classical proof is accepted only after a number of colleagues have read it and convinced themselves of the validity of each step. The multiplication $2 * 3 = 6$ is an acceptable step, but a proof consisting of 10^{31} such steps is not. For well-known computer results such as the Four Color Theorem (cf. [1, 2]) and the non-existence of the projective plane of order 10, this is the core of the problem: the outline of the proof was known long before the computational proof was finished. The tremendous amount of dull repetitive work, far too much for an ordinary work station, gives a kind of proof that is rather reluctantly received.

Figure 2. N.G. de Bruijn originated in 1968 the idea of machine verified proofs, using the Automath languages. Courtesy Birkhaüser inc.

4.1. Projective planes of order 10

A projective plane of order 10 is a configuration consisting of a set of 111 *points* and just as many subsets of size 11 of the point set, called *lines*, such that each pair of distinct points is on exactly one line, and each pair of distinct lines meets in precisely one point.

The theorem by C.W.H. Lam, L.H. Thiel and S. Swiercz (cf. [12, 13]) says that such a projective plane does not exist. At the time they finished the non-existence proof on computer, coding theoretic arguments had led to the observation that if a projective plane of order 10 existed, it would have at least one of 66 different well-specified subconfigurations on 19 points. Very crudely, the proof of Lam c.s. consists of an exhaustive search for possible extensions of each of these 66 subconfigurations to a projective plane of order 10.

So here each error in the proof could have blocked a conceivable road to finding a projective plane. In their announcement [13], the authors are extremely careful in formulating their result. I quote:

This note reports the result of a computer search for 19-point configurations, which, when taken together with previous results, implies that a plane of order 10 does not exist.

Part of the proof is carried out on one of the fastest supercomputers available at the time (1989): the CRAY-1A. Months of computer time have been used on that machine. Thus, the search cannot easily be repeated by a colleague (at least not as of 1995). Also, the slim chance of a shocking result does not motivate other researchers to repeat the effort. After all, the odds are high that the projective plane of order 10 does not exist indeed!

4.2. Oracles

The possibility to repeat the computational proof came up as a criterion for acceptance. Similarly to the case of a classical proof, it is of particular importance that colleagues involved in the validation procedure of a computational proof will be able to perform their verifications on their own machines with relative ease. In particular, if the result of a computation can be verified in a way that has little or (even better) nothing to do with the computation itself, and is much quicker to verify than the original computation, this condition is satisfied. A simple but typical instance of an independent verification is the factorization of large numbers like RSA-129.

Theorem (A.K. Lenstra et al.) RSA-129:

$$114381625757888867669235779976146612010218296721242362562561842 9$$
$$35706935245733897830597123563958705058989075147599290026879543541$$
$$=$$
$$349052951084765094914784961990389813341776463849338784399082057 7 *$$
$$327691329932667095499619881908344614131776429679929425397982885 33$$

It took Lenstra et al. several months to factor this 129 digit number, whereas the verification of the result is a single multiplication of two large numbers. Borrowing the terminology from circles in which the study of non-deterministic polynomial time algorithms is popular, I will call this the *oracle function* of the algorithm: it delivers, in a way that is irrelevant to the user, a result that we can check ourselves for correctness. In everyday use of computer algebra systems, the oracle function is not always so blatantly prominent.

4.3. Algorithms

Of course, in a computational proof, the possibility of hardware failures also needs to be considered. As we have recently seen in the case of the pentium

chip, even the basic arithmetic of a (new) processor may be erroneous. Also, the chance of a spontaneous error in computer arithmetic, for example caused by a speck of dust, is small but not inconceivable. Lam c.s. mention a detection of such an error on the CRAY. This is another reason why repetition or, even better, repeatability of a computational proof on different processors should be a necessary condition.

But what holds for hardware and basic arithmetic, also holds for software, for the implementation of an algorithm. If we accept the result of a multiplication on computer, then why should we not accept more complicated programs? Here too, different incarnations of the algorithm will enhance the acceptability of a computational proof if they provide the same output.

4.4. The Buchberger algorithm

In order to go into somewhat greater depth regarding the use of software, I will discuss the use of one of the key results of computer algebra: the Buchberger algorithm. This algorithm takes as input a system of polynomial equations in several unknown, and outputs an equivalent system of equations, from which the solutions can be read off almost immediately (in general after use of a factorization algorithm for polynomials in a single variable). The output is often called a *Gröbner basis*. This 'normal form' for polynomial equations has many useful applications. There are implementations of the Buchberger algorithm in the systems Bergman, CoCoa, Felix, Ganith, GB, KAN, Macaulay, Maple, Mathematica, Reduce, Saclib2, Singular,...

The Buchberger algorithm is a beautiful generalization of both the gcd algorithm for polynomials in a single variable and Gauss elimination for linear equations in several unknown. But here I do not want to go into the theory of the Buchberger algorithm; that has been done quite frequently lately; see for instance [6]. I would rather deal with its use in two examples from my own experience. One of the reasons why the algorithm has become such a great success, lies in the fact that many mathematical problems can be formulated as solving polynomial equations.

5. THE ICOSAHEDRAL GROUP

For the first example, consider the well-known icosahedron (see also figure 3). The icosahedral group is by definition the group of all symmetries (or, if you prefer, isometries) of the icosahedron. I will show how, using the Buchberger algorithm, we can transform the purely geometric description into algebraic data, namely a matrix form for each of three reflections generating the group. This in turn can be used for an algebraic description of the points of the dodecahedron.

All reflections leaving the icosahedron invariant, are of the same type: they reflect in a hyperplane as indicated in figure 4.

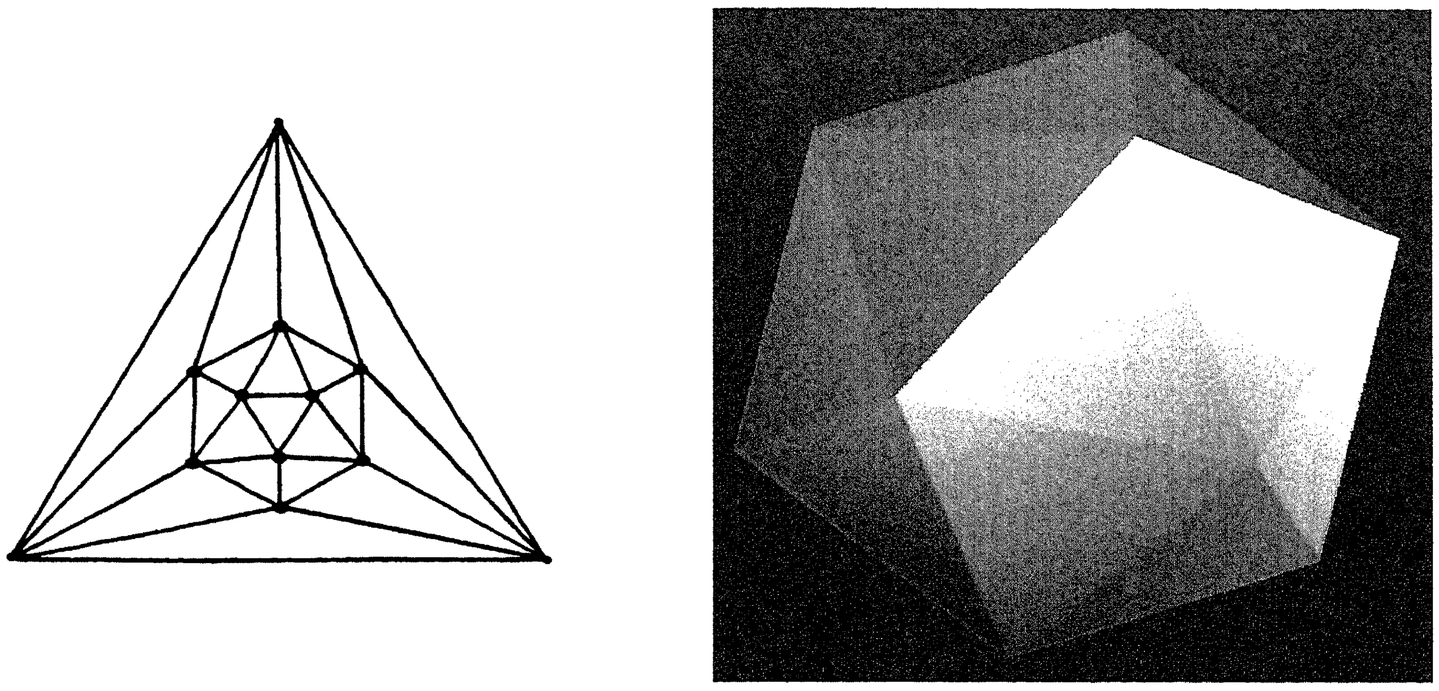


Figure 3. The icosahedron: as a graph (left) and 3D (right).

We can now choose three reflections x, y and z in such a way that the icosahedral group is generated by them. To this end, we take the angles between the various pairs of reflecting hyperplanes to be $60°$ for x and y, $90°$, for x and z, and $36°$ for y and z.

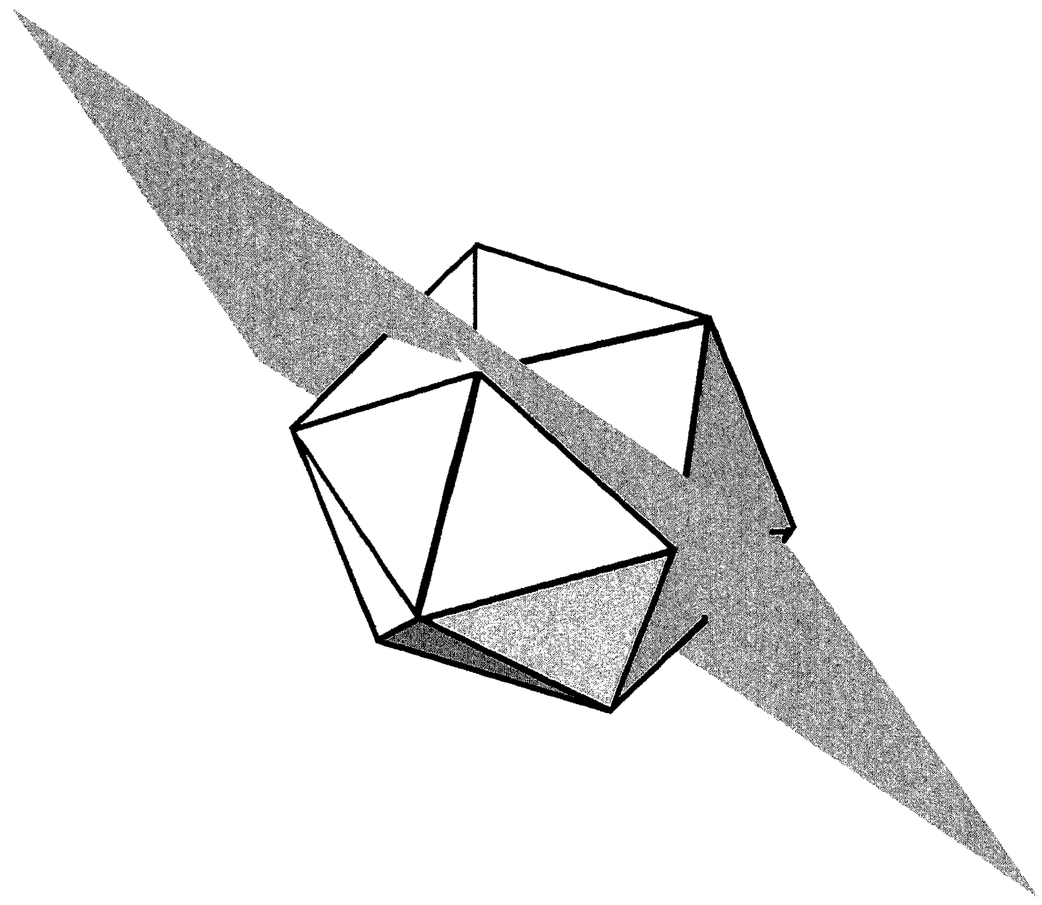


Figure 4. Reflections leaving the icosahedron invariant.

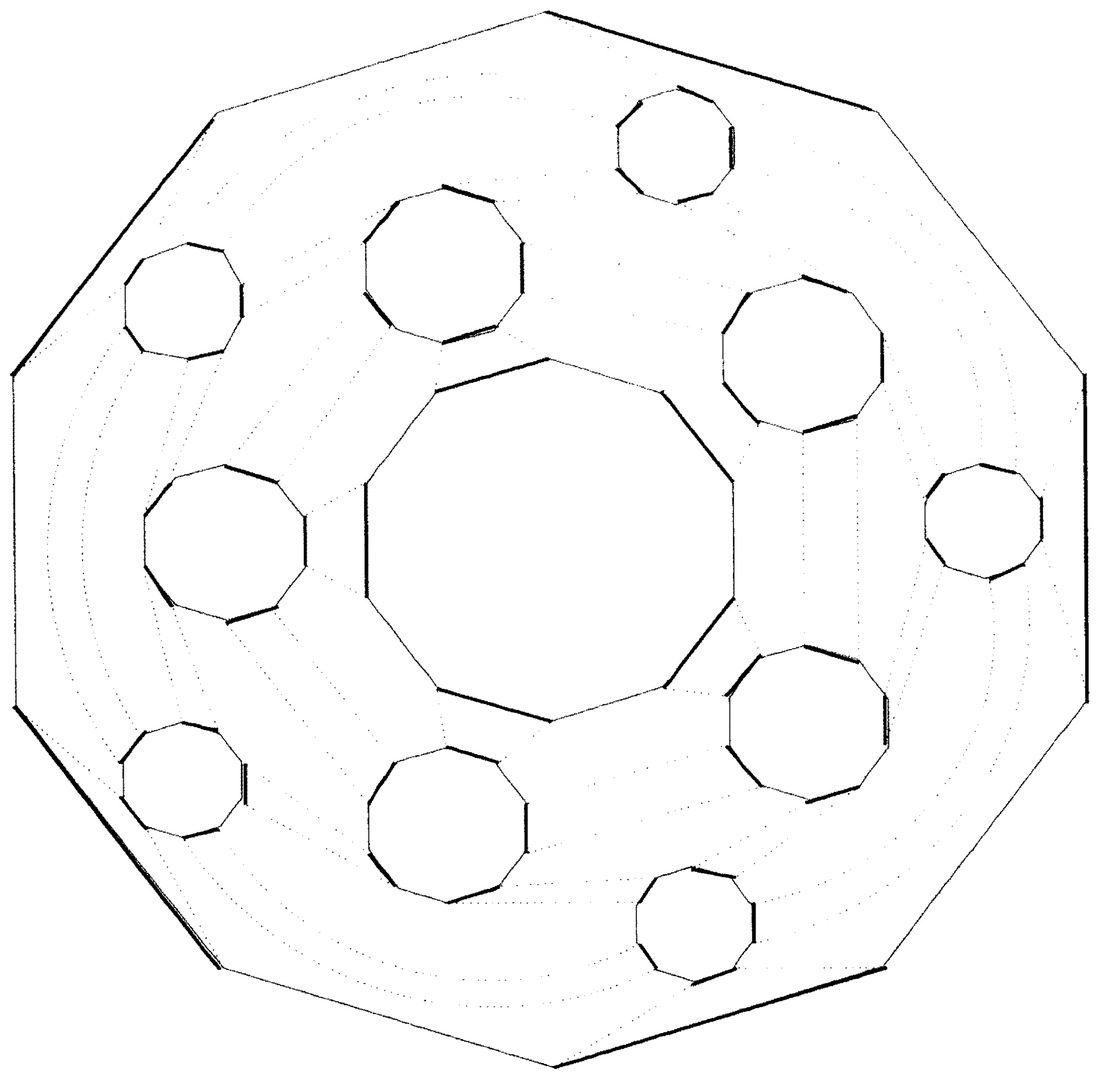


Figure 5. A Cayley graph of the icosahedral group.

We then have the following relations for x, y, and z:

$$x^2 = y^2 = z^2 = 1.$$

$$(xy)^3 = (yz)^5 = (xz)^2 = 1.$$

The first line expresses the fact that the reflections x, y, and z each have order 2. The second line holds because the product of two reflections is a rotation along twice the angle between the corresponding reflecting hyperplanes.

According to Coxeter the relations given are *defining* relations for the icosahedral group. To elaborate on this, we shall from here on view the icosahedral group as the abstract group generated by abstract elements x, y, and z subject to the relations given above. To see that this definition of the icosahedral group coincides with the former, one can invoke a procedure known as 'Todd-Coxeter' enumeration, which results in a so-called Cayley graph. From a free construction of this graph, in which each edge is labeled with one of the three reflections, figure 5 results.

Here, the labeling is as follows: a dotted line segment corresponds with x, an ordinary segment with y, and a fat segment with z.

The number of vertices of this graph is 120, the number of elements of the icosahedral group. These points can be identified (in a meaningful manner) with the elements of the group. By the way, the icosahedron itself can be recovered from this picture by fusing each of the twelve 10-gons to a point.

5.1. Embedding of the icosahedral group

But our goal is to show how, by use of the Buchberger algorithm, we can find orthogonal matrices for the elements of the icosahedral group. We can restrict ourselves to a search for matrices of the reflections x, y, z. Since they generate the whole group, every element can be written as a product of these (and all products are elements of the group).

Because x and z commute, we can choose them, without loss of generality, as follows:

$$x = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad z = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Thus, it remains to find the matrix $y = (y_{i,j})_{1 \le i,j \le 3}$. Here are some excerpts of a Maple program in which Buchberger's algorithm is called to solve the equations for the 9 coefficients of y deduced from the defining relations for the generators x, y, and z of the icosahedral group.

1. We load the linear algebra package, and input the 3 matrices x, y and z.

2. As a further preparation, we put the unknown entries of the matrix y in a list

   ```
   > vars := [y11,y12,y21,y13,y31,y22,y23,y32,y33]:
   ```

   and create the identity matrix of size 3:

   ```
   > idm := matrix(3,3,[[1,0,0],[0,1,0],[0,0,1]]):
   ```

3. We have written a routine `mkeq` that, given a matrix, puts its coefficients into a set. Using this routine, we put the equations resulting from $y^2 = 1$ into the set `eqy`.

   The expression `evalm` stands for 'evaluate as a Matrix'. When executed, it will write out the formal object `y ^ 2` as a matrix. The expression `idm` stands for the identity matrix of size 3.

   ```
   > y2 := evalm(evalm(y^2) - idm): eqy := mkeq(y2);
   ```

   $$eqy := \{y31\ y12 + y32\ y22 + y33\ y32, y21\ y13 + y22\ y23 + y23\ y33, y31\ y11 + y32\ y21 + y33\ y31, y21\ y11 + y22\ y21 + y23\ y31,$$

$y11\ y13 + y12\ y23 + y13\ y33, y11\ y12 + y12\ y22 + y13\ y32,$
$y12\ y21 + y22^2 + y23\ y32 - 1, y13\ y31 + y23\ y32 + y33^2 - 1,$
$y11^2 + y12\ y21 + y13\ y31 - 1\}$

4. Similarly for the equations coming from $(xy)^3 = 1$. Here the relation is rewritten to $xyx = yxy$ so as to keep the degree of the polynomials as low as possible. The set of equations is called `eqxy`.

```
> xyx := evalm(  x &* y &* x): yxy:= evalm( y &* x &* y):
> eqxy := mkeq(evalm(xyx - yxy)):
```

5. We continue with the equations from $(yz)^5 = 1$, calling the result `eqyz`.

6. Because y is a reflection, it has trace 1. Also, the matrix xy of order 3 should have trace 0 (the order cannot be 1 as x and y are distinct). This gives two linear equations for the coefficients of y. They can be obtained as follows:

```
> lineqs := {trace(evalm(y ) ) -1, trace(evalm(x &* y ) )};
```

$lineqs := \{\ y11 + y22 + y33 - 1, -y11 + y22 + y33\ \}$

Theoretically, these linear equations are superfluous. But experience tells us that whenever possible, linear equations should be added to reduce the complexity of the problem as much as possible.

7. The orthogonality conditions are also being translated into equations:

```
> yo := evalm( y &* transpose(y) - idm): eqo := mkeq(yo);
```

$eqo := \{y21^2 + y22^2 + y23^2 - 1, y31^2 + y32^2 + y33^2 - 1,$
$y11^2 + y12^2 + y13^2 - 1, y21\ y11 + y12\ y22 + y13\ y23,$
$y31\ y11 + y12\ y32 + y13\ y33, y21\ y31 + y32\ y22 + y23\ y33\}$

8. We now collect all equations found so far:

```
> eqs := eqy union eqxy union eqyz union lineqs union eqo;
```

$eqs := \{y11 + y22 + y33 - 1, y21^2 + y22^2 + y23^2 - 1,$
$y31^2 + y32^2 + y33^2 - 1, y11^2 + y12^2 + y13^2 - 1,$
$\ldots \quad \ldots \quad \ldots$
$y33 + y13\ y31 - y23\ y32 - y33^2, y11 + y11^2 - y12$
$y21 - y13\ y31,$
$-y12 + y11\ y12 - y12\ y22 - y13\ y32, -y11 + y22 + y33\}$

9. It is time to load the Gröbner basis package, and invoke the Buchberger algorithm. Here, the ordering of the variables in the list `vars` plays a role.

10. The Gröbner basis found by Maple is

$$\left\{ \begin{array}{c} 2y_{11} - 1, \\ y_{12} + 4y_{33}y_{32}y_{31} + 2y_{31}y_{32}, \\ y_{21} + 4y_{33}y_{32}y_{31} + 2y_{31}y_{32}, \\ y_{13} - y_{31}, \\ 2y_{31}^2 + y_{33} - 1, \\ 2y_{22} + 2y_{33} - 1, \\ y_{23} - y_{32}, \\ -1 + 4y_{32}^2, \\ -2y_{33} + 4y_{33}^2 - 1 \end{array} \right\}$$

Notice that the matrix y is symmetric. This we could have known in advance as it has order 2 and is orthogonal.

11. From the upper triangular structure of this Gröbner basis we can read off the general form of a solution. The last equation is quadratic in the single unknown y_{33} and so can easily be solved.

```
> gbo[9];
```

$$-2\,y33 + 4\,y33^2 - 1$$

```
> solve(");
```

$$\frac{1}{4} + \frac{1}{4}\sqrt{5}, \frac{1}{4} - \frac{1}{4}\sqrt{5}$$

```
> y33 := "[1];
```

$$y33 := \frac{1}{4} + \frac{1}{4}\sqrt{5}$$

In general, we need to factor the polynomial. Each irreducible factor then describes the algebraic numbers which the variable can take as values in an algebraic extension field. In our case the quadratic equation for y_{33} gives directly that, up to algebraic conjugates, y_{33} equals $-\cos(4\pi/5) = \frac{1+\sqrt{5}}{4}$, a familiar number in the context of the icosahedron.

12. By successively solving the equations one by one in the opposite order to which they are listed, we obtain the complete solution for y. Let me describe the next step:

```
> gbo[8];
```

$$-1 + 4\,y32^2$$

```
> solve(");
```

$$\frac{1}{2}, \frac{-1}{2}$$

```
> y32 := "[1];
```

$$y32 := \frac{1}{2}$$

13. Continuing this way, we find, up to algebraic and matrix conjugates, the unique solution:

$$y = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} - \frac{1}{4}\sqrt{5} & -\frac{1}{4} + \frac{1}{4}\sqrt{5} \\ -\frac{1}{4} - \frac{1}{4}\sqrt{5} & \frac{1}{4} - \frac{1}{4}\sqrt{5} & \frac{1}{2} \\ -\frac{1}{4} + \frac{1}{4}\sqrt{5} & \frac{1}{2} & \frac{1}{4} + \frac{1}{4}\sqrt{5} \end{bmatrix}.$$

Thus, we have not only found a solution y, but also know that, up to certain conjugacies, the solution is unique.

5.2. Application

I will show how, as a byproduct, we can find the coordinates of the 12 vertices of an icosahedron: choose a vector h fixed by the reflections y and z (for example $h = [-\frac{1+\sqrt{5}}{2}, 1, 0]$). Then repeated application of x, y, and z to h will produce a set (a so-called orbit of the group) consisting of the 12 vertices.

$$B = \left\{ \left[1, 0, \frac{1}{2} + \frac{1}{2}\sqrt{5}\right], \left[-1, 0, \frac{1}{2} + \frac{1}{2}\sqrt{5}\right], \left[0, -\frac{1}{2} - \frac{1}{2}\sqrt{5}, 1\right], \right.$$
$$\left[0, -\frac{1}{2} - \frac{1}{2}\sqrt{5}, -1\right], \left[0, \frac{1}{2} + \frac{1}{2}\sqrt{5}, 1\right], \left[1, 0, -\frac{1}{2} - \frac{1}{2}\sqrt{5}\right],$$
$$\left[0, \frac{1}{2} + \frac{1}{2}\sqrt{5}, -1\right], \left[-\frac{1}{2} - \frac{1}{2}\sqrt{5}, -1, 0\right], \left[-\frac{1}{2} - \frac{1}{2}\sqrt{5}, 1, 0\right],$$
$$\left.\left[\frac{1}{2} + \frac{1}{2}\sqrt{5}, 1, 0\right], \left[\frac{1}{2} + \frac{1}{2}\sqrt{5}, -1, 0\right], \left[-1, 0, -\frac{1}{2} - \frac{1}{2}\sqrt{5}\right]\right\}$$

To find the edges of the icosahedron one can proceed likewise.

5.3. Conclusion

Summarizing, we have two results. In the first place, we have a concrete presentation of the icosahedron and its group. To prove the correctness of this presentation we only have to check that a system of polynomial equations has a certain solution. Here, the Buchberger algorithm played the role of an oracle for finding it.

In the second place we have found that, in a certain sense, the solution is unique. (To be more precise: up to algebraic and matrix conjugation, there is a single matrix representation of the icosahedral group in which x, y, and z become reflections.) To prove the correctness of this assertion, it may seem that verification of the full arithmetic in Buchberger's algorithm is necessary—similar to the situation of the projective plane of order 10.

But Buchberger's theory gives a tool to see the Gröbner basis as an oracle. To this end, some more a priori arithmetic is necessary (or rather, some more storing of byproducts), in the same vein as the extended Euclidean algorithm needs more (storage) than the usual gcd computation. (When determining the gcd of two polynomials f and g it is not hard to deliver two polynomials a and b such that the gcd is $af + bg$. Now the verification that a given polynomial d is the gcd is nothing but the check that $d = af + bg$ and $d|f$ and $d|g$. This is shorter than the gcd computation itself.) Just like in these two well-known examples, there exists an 'extended Buchberger algorithm', which gives as extra output a way in which the Gröbner basis elements can be written as a linear combination of the input equations. Due to the extra output, the verification that, for a given system of equations, the output is indeed a Gröbner basis, is brought back to an exercise in standard polynomial arithmetic.

It is unfortunate that most commonly used computer algebra systems do not have standard facilities for the execution of the extended Buchberger algorithm. (Macaulay and Singular have a 'lift' command that does the job.) This omission may point to a somewhat all too practical attitude with which the general purpose packages are being used: the results are being accepted in gratitude, but the validity of their outcome is not always questioned to the extent that one would need for a mathematically acceptable proof.

6. A LARGER EXAMPLE

In the example just given, the proof that the embedding exists and is unique (in a certain sense), can be given in many other ways (for instance by use of classical character theory of groups). An important reason for presenting it in the context of a Gröbner basis computation, is that it is representative of cases where no other method is available for achieving a comparable result.

6.1. Kostant's conjecture

One such instance is Kostant's conjecture, which asserts that certain finite groups occur as subgroups of certain Lie groups. In the hardest case, the Lie group involved is the one of type E_8. In order to read on, you need not know more about this Lie group than the fact that it is a 248-dimensional variety of square matrices of size 248, whose group multiplication is the ordinary matrix multiplication. Let us call this group H (abbreviating 'haystack').

According to Kostant's conjecture, this very large group should have a very tiny group as a maximal closed Lie subgroup. This tiny group is the simple group of size 113460, and is known as the fractional linear group over the field of order 61. It is usually denoted by $L(2,61)$, but here, we will denote it by N (abbreviating 'needle').

Thus we are facing the question whether the tiny group N embeds in the large group H, and if so, how.

6.2. The solution

Some ten years ago R.L. Griess Jr. and I brought the problem back to a system of polynomial equations. The method we used, although more delicate, is comparable to the approach we described for the icosahedral group: the needle N is generated by three elements u, t and w satisfying the following defining relations:

$$\begin{aligned} u^{61} = t^{30} = 1, &\qquad tut^{-1} = u^4, \\ w^2 = 1, &\qquad wtw = t^{-1}, \\ (uw)^3 = 1, &\qquad wu^2w = t^{-1}u^{-2}wu^{30}. \end{aligned}$$

The matrices for u and t were easy to determine by use of some Lie theory. The coefficients of the third matrix, w, could be described as rational functions of 9 parameters. The equations that could be directly derived from the defining relations were too large to handle. Therefore, more complicated computations were set up, which made use of projections onto t-eigenspaces.

This led to a system of 57 equations in 9 unknown. Each equation had about 9 monomials. All by itself, it is nothing particular of a problem that it can be put into a system of polynomial equations. Solving the system of equations is another ball game, though. Around 1986, at the time the larger general purpose computer algebra systems came about, I tried to solve these equations in vain (using Macaulay and Maple).

About four years ago, together with B. Lisser, I ventured another try. After having made the preparations for solving the set of polynomial equations, I found a way to dodge the polynomial equations by solving a system of more than 1000 *linear* equations in 248 unknown. This system turned out to be solvable with Gauss elimination in several computer algebra systems.

When I tried to solve the large (heavily overdetermined) system of linear equations, the diagonal that appeared due to Gauss elimination, crept on towards the last column, until it stopped at the one but last, and stayed there for all of the overdetermining equations that were to follow. At that moment I was very sure I had found the needle N in the haystack H, and a unique one at that. Nevertheless, I performed all the necessary verifications to establish that my findings were correct: the *computational proof* was clearly presented (cf. [5]).

But, in a way, I was *counting* too heavily on the convincing power of the computer. Only little later I would have to *account* for what I had done. When it came to publication, my colleagues were quite skeptical regarding the computer computations. I had to pay the price for computations that I could not do by hand: the price that the computations could not be accepted as proof. Or, to put their reactions into a milder perspective: I was asked to specify under which circumstances computer calculations are acceptable as (part of) a proof, in particular, if they can no longer be checked by any person by hand.

6.3. Computer calculations as part of a proof

Just as I have done for the icosahedron, I will address the proofs for existence and for uniqueness separately.

Existence

To verify that the three matrices u, t and w satisfy the defining relations, we only have to perform standard calculations. Once we have chosen a suitable coefficient domain for our computations, the necessary matrix multiplications can be carried out in any one of the special purpose packages GAP, MAGMA, and LiE. Each matrix multiplication will take a few seconds, but that is quite an acceptable time span, even for interactive work.

In this manner, each colleague can verify that u, t, and w generate a subgroup of the group of all invertible square matrices of size 248, which is isomorphic to N. In order to finish the proof of Kostant's conjecture in this case, we still need to verify that each of u, t, and w belongs to the group H. Again, this verification does not require anything beyond standard arithmetic.

Uniqueness

Here we were lucky. Because of linearity of the system of equations eventually used, uniqueness can be either derived from the (straightforward) repetition of the Gaussian elimination or from an LUP decomposition of the original system.

If we would have had to resort to the system of 57 polynomial equations found earlier, a uniqueness proof using the extended Buchberger algorithm

Definition Let K be a field.
1. An *associative algebra* over K is a tuple $(A, +, 0, -, *, \cdot)$, such that $(A, +, 0, -, *)$ is an associative ring, $(A, +, 0, -, \cdot)$ is a vectorspace over K, and $\forall \lambda \in K, x, y \in A : \lambda \cdot (x * y) = (\lambda \cdot x) * y = x * (\lambda \cdot y)$.
2. A *Lie algebra* over K is a tuple $(A, +, 0, -, [], \cdot)$, such that $(A, +, 0, -, \cdot)$ is a vectorspace over K, [] is a bilinear map, $\forall x \in A : [xx] = 0$ and $\forall x, y, z \in A : [x[yz]] + [y[zx]] + [z[xy]] = 0$.

Corollary Let $(A, +, 0, -, *, \cdot)$ be an associative algebra over K.
Define a binary map [] as follows $[xy] =_d x * y - y * x$.
Then $(A, +, 0, -, [], \cdot)$ is a Lie algebra over K.
Proof
1. [] is bilinear, because $[(\lambda \cdot x)y] = \lambda \cdot (x * y) - \lambda \cdot (y * x) = \lambda \cdot [xy] = [x(\lambda \cdot y)]$
and $[(x + y)z] = x * z + y * z - z * x - z * y = [xz] + [yz]$
and $[x(y + z)] = x * y + x * z - y * x - z * x = [xy] + [xz]$.
2. $[xx] = x * x - x * x = 0$.
3. $[x[yz]] + [y[zx]] + [z[xy]] = 0$ by computation#.

```
> noncom x,y,z;
                                        x, y, z noncom
> procedure br(a,b);a*b-b*a;
                                        procedure br
> br(x(1),br(y(1),z(1)))+br(y(1),br(z(1),x(1)))+br(z(1),br(x(1),y(1)));
                                        0
```

Computation session in Reduce to prove the Jacobi identity

Figure 6. Elaborate mathematical proofs rely increasingly on the use of Computer Algebra.

would have been desirable. But the usual version of this algorithm was already infeasible at the time. In the meantime, eight years after our first attempt, the system of 57 polynomial equations has been solved twice, first with Macaulay, later with Singular; in both cases the same (unique) solution was found, which of course coincided with the solution of the linear equations.

In summary, although matrices are involved of size larger than we can conveniently deal with by hand, the arithmetic carried out with up-to-date software is so standard, that they can be viewed as acceptable (parts of) proof. The requirement of repeatability is met.

6.4. Conclusion of the large example

In the above discussion I left out some aspects which are worth mentioning.

For instance, I referred to choosing a suitable 'coefficient domain'. Kostant's conjecture concerns Lie groups. Hence, it is formulated for the co-

Simple groups L having a central extension that can be embedded in the complex Lie group of exceptional type X_n	
X_n	L
G_2	Alt_5, Alt_6, $L(2,7)$, $L(2,8)$, $L(2,13)$, $U(3,3)$
F_4	Alt_7, Alt_8, Alt_9, $L(2,25)$, $L(2,27)$, $L(3,3)$, ${}^3D_4(2)$, $U(4,2)$, $O(7,2)$, $O^+(8,2)$
E_6	Alt_{10}, Alt_{11}, $L(2,11)$, $L(2,17)$, $L(2,19)$, $L(3,4)$, $U(4,3)$, ${}^2F_4(2)'$, M_{11}, J_2
E_7	Alt_{12}, Alt_{13}, $L(2,29)^?$, $L(2,37)$, $U(3,8)$, M_{12}
E_8	Alt_{14}, Alt_{15}, Alt_{16}, Alt_{17}, $L(2,16)$, $L(2,31)$, $L(2,41)^?$, $L(2,32)^?$, $L(2,49)^?$, $L(2,61)$, $L(3,5)$, $Sp(4,5)$, $G_2(3)$, $Sz(8)^?$

efficient domain of the complex numbers. The data on the finite group N however make it possible to realize all numbers involved as algebraic numbers. In theory, the exact arithmetic of algebraic numbers on computers is possible, and is in fact one of the major raisons d'être for computer algebra. But computations regarding square matrices of size 248 are not feasible when the coefficients are algebraic numbers of considerable size.

Therefore, we have chosen for reduction modulo a suitable prime number p. Due to some classical mathematical reasoning, it is necessary and sufficient for the embedding of N in H to solve the problem of finding matrices for u, t, and w over coefficients that are integers modulo p.

Besides simplification of the calculations, the technique of reduction modulo a prime number had another good consequence. It got J.-P. Serre interested, who recently produced a computer free proof of the embedding of N in H. He used reduction modulo the prime 61, which requires a much more intricate argument for lifting N back to H, but has the advantage that the subgroup N is known to exist (in the version of H mod 61) from the theory of groups of Lie type. By the way, Serre's proof does not give uniqueness of the embedding of N in H.

To end this example, I would like to mention that the work on Kostant's conjecture is part of a much bigger programme, namely to determine all maximal finite subgroups of the exceptional Lie groups. In this classification, only a few open problems are left. In the table above, question marks indicate which embeddings are still unpublished at the time of writing.

The table is taken from [7]; there however, the group $L(2,41)$ is erroneously left out. In the table L is always a finite simple group and G an exceptional complex simple Lie group (one of G_2, F_4, E_6, E_7, E_8). We provide a twofold interpretation of this table.

1. If L appears on the line of G in the table, then it has a finite central extension that is embeddable in G, with a possible exception for five question marks '?'.

2. If L appears neither on the line of G nor on a line above it, then no finite central extension of L embeds in $G(\mathbf{C})$.

One of the five question marks appears with the group $L(2,41)$ of fractional linear transformations over the field of 41 elements. Very recently (April 2, 1995) Serre announced a computer-free proof of the embedding of this group in H, to which Griess and A.E.J. Ryba reacted by announcing a computational proof in the making for the embedding of $L(2,32)$. Thus, serious mathematics sometimes has the likings of a correspondence chess game between G. Kasparov and a group of computer-chess players.

7. INTEGRATION

Now that we have gone over some of the features of the new appearances of a proof, I want to add a few words on their interaction. With regard to this topic, I have once heard a logician express the ideal of having all mathematics be verified by proof checkers. If this would imply that the computer algebra systems should account for each arithmetic step in their executions of algorithms, giving a deduction of it which can be input to a proof checker, I am not convinced it is the right goal.

First of all, it does not bring about any pragmatic help for the usual mathematical activities. Secondly, it is not feasible to make any mathematical progress on this basis, simply because proof checkers cannot perform arithmetic with ease and/or speed comparable to computer algebra systems. In the formal proof that 13 is a prime, we had to come up with axioms like $13 = 6 * 2 + 1$ and $13 = 4 * 3 + 1$ in order to keep the number of lines to reasonable length. A proof of an arithmetic equality like any of these two in LEGO comes down to writing out both hand sides as the 13-th successor of 0 within the natural numbers. So this would not be a feasible approach to proving that a certain 31 digit number is prime. (A much more realistic approach would be to prove correctness of the usual number arithmetic, recognizing strings of digits as numbers and next to use meta-theorems; but for simplicity I will overlook this possibility here, especially since eventually the arithmetic is bound to be delegated to software better suited for computations than proof checkers.)

I would rather favour the point of view where proof checkers will accept identities coming from computer algebra systems. Some experiments in this direction have shown that at least this approach is feasible: from the proof checker LEGO, an expression has been sent off for simplification to the computer algebra system REDUCE; the resulting equality between the

input expression and the output expression has been fed into LEGO as an axiom. By combining the results of computer algebra work with proofs in proof checkers in this way, a much more powerful tool for mathematics is being created than any of the two can offer individually.

8. CONCLUSION

Having stressed repeatability and verifiability of a computational proof, I might have given the impression that the validity of a proof would be quantifiable. To refute this, consider the following thought experiment. Of an explicitly given number n of 31 digits, say, the assertion is being made that it is a prime number. As a proof of this assertion, a long chain of computations is presented. However, an error occurs in one of these computations (so in fact the chain of computations is *not* a proof).

Now suppose five colleagues peruse this erroneous proof independently, leaving a very slim chance ϵ, say $\epsilon = 10^{-15}$, that the error remains undetected.

I have already mentioned that arithmetic on a computer is not 100% reliable. But it is quite likely that, using independent repetitions, we arrive at a likelihood of more than $1-\epsilon$ that the error in the long chain of computations is found, regardless of whether the proof is written up as a formal proof or as a computational proof.

The point I am trying to make is that there are very short probabilistic arguments that, after verification, give a likelihood of at most ϵ that the assertion is wrong. One such a probabilistic proof for the assertion that n is prime, comes from Solovay and Strassen (cf. [3]), and makes use of the following result: For $n \in \mathbf{N}$, $n > 1$, n odd:

$$n \text{ is prime}$$

$$\Leftrightarrow$$

$$\forall\, k \in \mathbf{N},\ 0 < k < n,\ \begin{cases} \gcd(n,k) = 1 & \text{and} \\ k^{\frac{n-1}{2}} \equiv \left(\frac{k}{n}\right) & (\mathrm{mod}\ n), \end{cases}$$

where $\left(\frac{k}{n}\right)$ is the Jacobi symbol.

If n is not prime, then the likelihood that the technical condition above holds for a random k between 1 and n is at most $1/2$. Hence, the likelihood that n is not prime and that the condition holds for 50 random choices of k, is at most 2^{-50}, in particular less than $\epsilon \approx 10^{-15}$. The verification of the technical condition for a single k is extremely fast.

Therefore we can, with the probability of an error which is smaller than the likelihood of a non-detected error by our five colleagues, establish that the assertion is correct by means of a relatively short computation. Still I expect that only few mathematicians will accept this probabilistic argument as a real proof.

By use of this paradox I wanted to illustrate that a reliability estimate all by itself does not *count* for the notion of proof; we shall have to take the human, esthetic standards and values into *account*.

ACKNOWLEDGMENTS

I am very grateful to H.P. Barendregt, N.G. de Bruijn, H. Elbers, F.G.M.T. Cuypers, L.J. van Gastel, H.J.M. Sterk, H.A. van der Vorst for their valuable input.

The topic of the computer influence on mathematical proofs is also dealt with in [3], and various reactions triggered by it (among which [11]).

This text is a translated version of the closing address given by the author at the 31st Nederlands Mathematisch Congres, Groningen, April 21, 1995 and will also be published in *Nieuw Archief voor Wiskunde* (March 1996).

REFERENCES

1. K. APPEL, W. HAKEN (1976). Every planar map is four-colorable. *Bull. A.M.S. 82*, 711-712.
2. K. APPEL, W. HAKEN (1989). Every planar map is four colorable. *Contemporary Math., Vol. 98*, AMS, ISBN 0-8218-5103-9.
3. L. BABAI (1994). Probably true theorems, cry wolf? *Notices AMS 41(5)*, 453-454.
4. M. BEZEM, R. BOL, J.F. GROOTE (1995). *Formalizing Process Algebraic Verifications in the Calculus of Constructions*, Report CS-9502, Dept. Math. & CS, Eindhoven University of Technology.
5. A.M. COHEN, R.L. GRIESS JR., B. LISSER (1993). The group $L(2,61)$ embeds in the Lie group of type E_8. *Comm. Algebra 21*, 1889-1907.
6. D. COX, J. LITTLE, D. O'SHEA (1992). *Ideals, Varieties, An introduction to Computational Algebraic Geometry and Commutative Algebra*, Undergraduate Texts in Math., ISBN 3-540-97847-X, Springer-Verlag, Berlin.
7. A.M. COHEN, D.B. WALES (1993). Finite simple subgroups of semisimple complex Lie groups—a survey. To appear in *Proceedings of Groups of Lie type and their Geometries*. W.M. KANTOR (ed.), Como.
8. J.F. GROOTE, J. VAN DE POL (1993). *A Bounded Retransmission Protocol for Large Data Packets*, Preprint 100, Dept. Phil. Utrecht University.
9. L. HELMINK, M.P.A. SELLINK, F. W. VAANDRAGER (1994). *Proof-Checking a Data Link Protocol*, CWI Report CS-R9420, Amsterdam.
10. H. HRENNIKHOFF, D. MCHENRY (1990). see J.T. Oden's article in A history of scientific computing. S.G. NASH (ed.). ACM Press, New

York, 152–166. (ISBN 0-201-50814-1)

11. J. HORGAN (1993). Trends in Math., The death of proofs. *Scientific American, 74.*
12. C.W.H. LAM (1991). The search for a finite projective plane of order 10. *Amer. Math. Monthly.*
13. C.W.H. LAM, L.H. THIEL, S. SWIERCZ (1989). The non-existence of finite projective planes of order 10. *Canad. J. Math. 41*, 1117–1123.