J. Symbolic Computation (2001) **32**, 3–22
doi:10.1006/jsco.2001.0455
Available online at http://www.idealibrary.com on IDEAL®

Electronic Communication of Mathematics and the Interaction of Computer Algebra Systems and Proof Assistants

HENK BARENDREGT[†§] AND ARJEH M. COHEN[‡¶]

[†]*Department of Computer Science, Nijmegen University, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands*
[‡]*Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Present day computer algebra systems (CASs) and proof assistants (PAs) are specialized programs that help humans with mathematical computations and deductions. Although several such systems are impressive, they all have certain limitations. In most CASs side conditions that are essential for the truth of an equality are not formulated; moreover there are bugs. The PAs have a limited power for computing and hence also for assistance with proofs. Almost all examples of both categories are stand alone special purpose systems and therefore they cannot communicate with each other.

We will argue that the present state of the art in logic is such that there is a natural formal language, independent of the special purpose application in question, by which these systems can communicate mathematical statements. In this way their individual power will be enhanced.

Statements received at one particular location from other sites fall into two categories: with or without the qualification "evidently impeccable", a notion that is methodologically precise and sound. For statements having this quality assessment the evidence may come from the other site or from the local site itself, but in both cases it is verified locally. In cases where there is no evidence of impeccability one has to rely on cross checking. There is a trade-off between these two kinds of statements: for impeccability one has to pay the price of obtaining less power.

Some examples of communication forms are given that show how the participants benefit.

1. Introduction

This paper is an introduction to the field of computer assisted mathematics in which CASs and PAs interact, and is aimed at people who are not familiar with one of these two topics. At the same time we have tried to make it interesting to people that are familiar with one of these fields. This paper continues ideas presented in Cohen (2000) and Barendregt (1997).

[§]E-mail: `henk@cs.kun.nl`
[¶]E-mail: `amc@win.tue.nl`

CURRENT SITUATION

In computer algebra systems a rich variety of abstract (but "computable") mathematical objects can be represented. For example there are

$$\begin{array}{rl}
\text{algebraic numbers} & \sqrt{-5} \\
\text{polynomials} & x^3 - 3x^2 + 2x - 1 \\
\text{rational functions} & \dfrac{x^2+1}{x^3 - 3x^2 + 2x - 1} \\
\text{integrals} & \displaystyle\int \frac{x^2+1}{x^3 - 3x^2 + 2x - 1} dx, \ \int \frac{dx}{\sqrt[3]{1+x^6}} \\
\text{permutations} & (1,2)(4,6,8,7) \\
\text{matrices} & \begin{pmatrix} x & 1-x \\ 7/x & 0 \end{pmatrix}
\end{array}$$

and we can perform on these objects a large repertoire of computable operations. The algorithms implemented in CASs are impressive in two ways: they rely on sophisticated mathematics and they have a good efficiency performance.

PAs provide support in dealing with mathematical deduction. These systems range from automated theorem provers to proof checkers. An automated theorem prover performs a proof search once it is given a logic, an axiom system and a goal to be proved. A proof checker on the other hand is given in addition a formalized putative proof and then checks its correctness in the given situation. In some areas of mathematics—such as elementary geometry—there are theorem provers that work well, but in general proof search is not feasible. At the other end of the spectrum proof checking is a boring enterprise, since proofs have to be provided in full detail by a human. For these reasons many PAs are situated somewhere in between the two extremes. These PAs are interactive: in a dialogue with the human user the proof is constructed.

CASs and PAs are impressive in different ways. CASs can construct interesting mathematical objects which are difficult to obtain otherwise. At present the collaboration between humans and PAs has not yet produced substantial new mathematical theories, but the potential is there. The interest of PAs lies in the capacity to formalize large parts of mathematics, including abstract non-computable objects and notions, obtaining a considerably increased level of rigor. Moreover these systems are used for the verification of small but essential pieces of software.

PROBLEMS

Both the CASs and the PAs also have their problems. In CASs one usually does not have a means to state side conditions needed for the validity of an equality. This can lead to incorrect results presented by a CAS. And then there are bugs in CASs—as so often is the case in big software systems—because of the high complexity of the underlying mathematics and the optimization used for the performance.

In the present day PAs one has a less developed library of computable functions. Since computations are also important for the proofs of theorems (we will give examples later), this is a limitation indeed. The reason for this underdevelopment is twofold.

In the first place not all logical systems are very apt at representing computations. And in the second place not much energy has been spent on the development of computable functions in those systems in which this is possible. These two reasons have a common origin. Computations are long "objects" (sequences of stepwise transformed expressions). If such an object is subject to a proof of correctness, then that proof tends to be even longer. We will pause a moment to sketch this tension between proofs and computations (or if you want between rigor and efficiency) for a period of over 2000 years into the present. In Section 4 we will see how a suggestion of Poincaré, as implemented in various type theories, is able to make a smooth interface between proving and computing.

AN ABSTRACT HISTORY OF MATHEMATICS: COMPUTATIONS VS. PROOFS

It is said that mathematics started at least 6000 years ago. What is meant by this statement is that bones from that period have been found on which there are carvings, indicating that a process of counting (in unary notation) took place. About 4000 years ago Sumerian mathematics used numbers represented in a (sexagesimal, i.e. base 60) positional notation and the Babylonians were able to compute with these and solve quadratic equations. Around 2500 years ago the Greeks made another quantum leap forward by introducing reason and proofs. This culminated in the work of Euclid (300 B.C.), summarizing large parts of then known mathematics in an axiomatic way. Although proofs are still considered as the essence of mathematics because of the implied rigor, at first they came at the price of not being able to handle computations well. For example in Euclid one can find an equality like $(x + y)^2 = x^2 + 2xy + y^2$ explained in geometrical terms, but never similar equations for $(x + y)^4$, for obvious reasons. Archimedes (287–212 B.C.) was an exception in his mastery of combining computations and proofs, but after him it took a long time until the two activities of mathematics, computing and reasoning, were fully integrated. One of the next steps came from the Persian mathematician al-Khwarizmi, who gave (around 825 A.D.) proofs that the elementary arithmetical operations on the decimal positional system (as taught in our elementary school) are indeed correct. But when Newton introduced in his Principia (1686) the calculus, he went through quite some effort to formulate it in geometrical terms in order to achieve the same rigor as Euclid. This difficulty did not hamper him in obtaining fine results. But when in the eighteenth century English mathematicians continued in this fashion, their progress was seriously hampered. In continental eighteenth century mathematics computations became a powerful tool culminating in the work of Euler. Not all his formal manipulations were correct, but Euler's intuition was strong enough to be a guide to what to accept and what not. It was only in the nineteenth century that a sufficiently rigorous foundation was given for calculus by Cauchy, Weierstrass and others. Since that period modern mathematics has flourished.

Once the foundations of calculus were satisfactory, one needed to deal and reason with infinite objects. In order to obtain a sufficient ontology and logical foundation for this purpose one had to wait untill the twentieth century, when formalisms such as set theory, (higher order) logic or type theory became available. We will come back to this later.

Although the tension between computations and proofs ended in the nineteenth century for the usual informal mathematics, the story continues since now parts of mathematics are done with the help of computers.

2. Computer Algebra Systems

EXPRESSIVENESS

In CASs, arithmetic with mathematical objects such as arbitrary length integers, polynomials, transcendental functions, permutations, groups, differential operators, vectors and matrices is standard. These systems work with expressions representing recursively built up data types. For example, the following gives a possible abstract grammar for expressions over a ring.

$\mathrm{nzd} := 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$	non-zero digits
$\mathrm{d} := 0 \mid \mathrm{nzd}$	digits
$\mathrm{n} := \mathrm{d} \mid \mathrm{nzd}\,\mathrm{n}$	numerals
$\mathrm{pv} := x \mid y \mid z$	primitive variables
$\mathrm{v} := \mathrm{pv} \mid \mathrm{pv}_{\mathrm{n}}$	(general) variables
$\mathrm{e} := 0 \mid 1 \mid \mathrm{v} \mid \mathrm{n}\,\mathrm{e} \mid \mathrm{e}{+}\mathrm{e} \mid \mathrm{e}{-}\mathrm{e} \mid \mathrm{e}{\times}\mathrm{e} \mid \mathrm{e}^{\mathrm{n}}$	(ring) expressions

Typical examples of such expressions are $x^2+2xy+y^2$ and $(x_0x_1x_2x_3+1)^{10}$. The first one is built up as follows: x and y are primitive variables, hence variables, hence expressions; but then $x^2, 2xy$ and y^2 are also expressions and finally $x^2+2xy+y^2$ is one too (we use the associativity law to be implicit about how the sum is formed); the second expression is built up similarly after we realize how the numeral 10 is constructed.

COMPUTATIONAL POWER

The standard arithmetical operations addition, subtraction, multiplication, and division, on mathematical objects as described in the Introduction, are available whenever defined. But many other operations are possible, ranging from simplification of expressions, factorization of polynomials and integration of transcendental functions to finding the reduced Gröbner basis of an ideal when given a set of generating polynomials for it. The reader is referred to any of the following authors; Cohen and Sterk (1999), Adams and Loustaunau (1994), Buchberger (1985), Buchberger and Winkler (1998), Cox and O'Shea (1992) and Eisenbud (1995), for details on Gröbner bases.

EFFICIENCY

The size of objects that can be stored and the speed of standard arithmetic greatly exceeds human capacity. For example, groups generated by permutations on thousands of letters can be efficiently analyzed. Fast algorithms for factorization are still being developed, enabling users to decompose rational univariate polynomials of degrees in the hundreds. Solving a system of polynomials by means of Gröbner basis techniques has become considerably more efficient over the last few years, but the extremely bad complexity does not allow for any optimism in the size of systems that can indeed be solved automatically.

RELIABILITY

CAS are not impeccable for two reasons. In the most commonly used systems, there is no mention of side conditions. Thus, when asking for the integral $\int x^a \mathrm{d}x$, you often receive as an answer $\frac{1}{a+1}x^{a+1}$, although this fails to hold for $a = -1$. The other reason is simply the existence of bugs. For example, although clearly

$$1 - \frac{1}{2 + e^{3\pi ix}} = \frac{1 + e^{3\pi ix}}{2 + e^{3\pi ix}},$$

according to Maple V, release 4, the definite integral on the interval $(0, 2)$ of the left-hand side equals 2 while the same integral of the right-hand side equals 0.

3. Logic and Type Theory

Aristotle gave a good description of the axiomatic method. In mathematics there are objects and properties. Objects are either defined (from other known objects) or primitive objects. Properties can either be proved (from other established properties) or be assumed axiomatically.

Computation deals with the manipulation of objects, proving with deriving properties. What should be emphasized is that definitions also play an important role in the formulation of properties.

FROM INFORMAL TO FORMAL MATHEMATICS

First we will give a global view of the ways in which informal mathematical statements can be formalized in logic and type theory.

INFORMAL MATHEMATICS

A typical statement in informal (i.e. the usual kind of) mathematics is the following.

THEOREM. *Given situation* Γ. *Then one has* A.

The fact that mathematics is informal does not mean that it is imprecise, but that it is open ended. The language in which it is formulated is informal, not given by a fixed grammar. Since informality can go together with precision one speaks about *informal rigor.* In spite of this it is customary in informal mathematics that the precision in the formulation of the results is higher than in the proofs.

LOGIC

In logic the statement in the above theorem becomes the following assertion.

$$\Gamma \vdash A \tag{1}$$

stating that from the assumptions Γ the statement A is provable. The statements Γ, A are formalized, but not always the proof[†].

The simplest logic that is adequate for reasonable parts of mathematics is *first-order* logic. Here one considers a domain of objects in which one is interested and quantification ("for all", "there exists") ranges over this domain.

[†]One often relies on informally proved metamathematical results; for example in predicate logic with equality one has $a = b \Rightarrow C[a] = C[b]$, for all contexts $C[\ ]$. Although this can be derived for each instance of $C[\ ]$ from the equality axioms, one does not do so, because such derivations become lengthy. Rather one informally proves the general statement by induction on the structure of $C[\ ]$ in the metalevel.

A reasonable amount of geometry or number theory can be developed in the axiomatizations of these theories in first-order logic (e.g. by Hilbert and Peano). More profound results are obtained by making connections with notions from other parts of mathematics, for example by studying geometric structures in the context of groups. If the other parts of mathematics can be formalized within first-order logic, then the richer theory can be formalized best by using *many-sorted* (first-order) logic (L_1). In this framework one considers several domains of objects (with quantification over one of these). A prime example is the theory describing vector spaces in which there is a domain for scalars and one for vectors.

But some theories cannot be formalized in a straightforward way within first-order logic. An example of the other way to obtain more profound results is to apply analytical methods to arithmetic (for example obtaining results about primes). For this one needs to work in a framework that is stronger than first-order logic. There are (at least) two different means by which to do this. The first one is the program of Bourbaki that formulates mathematics in set theory (ST), which on the one hand is a theory strong enough to do this and on the other hand capable of being formalized within first-order logic. (The many-sortedness is automatic, as we can describe many different universes (structures) within set theory.) In this way (1) is replaced by

$$\Gamma_{ST}, \Gamma \vdash_{L_1} A.$$

The second way is to use second- or higher-order (many-sorted) logic in which many mathematical notions can be defined. The difference with first-order logic is that now quantification may range over subsets of the domains (second-order logic; L_2) or families of subsets of the domains (third-order logic; L_3), etc. until higher-order logic (L_ω, the union of the L_n). Now (1) is replaced by

$$\Gamma \vdash_{L_2} A, \qquad \Gamma \vdash_{L_3} A, \ldots \qquad \text{or} \qquad \Gamma \vdash_{L_\omega} A,$$

depending on the logical strength that is needed. (Some authors have a different way of counting this hierarchy.)

There is still one aspect of mathematics that is not covered by the logical machinery so far. Suppose we tell you that for \$10 we can give you a machine that when turned on within 1 second answers “yes” if and only if the Riemann hypothesis is true and answers “no” otherwise. Then you may be willing to risk your \$10 and ask for the machine. What we do is that we give you two machines: one that always answers “yes” within 1 second and another one that always answers “no”. We reason that if the Riemann hypothesis is true, then the first machine is the right one, and if it is false, then the second machine is. But in any case we fulfilled our promise: we gave you a machine with the right performance. Those of you that find this a cheat are right. But this is exactly what Brouwer meant by his rejection of the general principle of the excluded middle. Another way of seeing his point is that there are predicates $A(x)$ in Peano arithmetic (PA) such that one has $\Gamma_{\mathrm{PA}} \vdash \exists x.A(x)$ but

$$\Gamma_{\mathrm{PA}} \nvdash A(0), \qquad \Gamma_{\mathrm{PA}} \nvdash A(1), \qquad \Gamma_{\mathrm{PA}} \nvdash A(2), \ldots.$$

This means that one can prove that a certain natural number exists, without being able to find a witness.

It may seem that if mathematics is based on intuitionistic rather than classical logic one can prove much less. Among others Gödel pointed out that one can have another view. By making the following “double negation translation” that essentially replaces atomic formulas by their double negation and disjunctions and existential formulas by their so-called weak versions

$$A \vee^o B \equiv \neg(\neg A \& \neg B)$$
$$\exists^o x.A \equiv \neg\forall x.\neg A$$

a classical mathematician does not notice anything, while now one can prove $A \vee^o \neg A$ within intuitionistic logic. In fact provability based on intuitionistic logic is stronger: now one can for example express strong existence $\exists x.Ax$, which is such that if it is provable, then one always can find a witness n such that An is provable. Or if one proves $A \vee B$, then one can find out a proof of one of the two.

TYPE THEORY

Type theory goes one step further by formalizing the proofs as well. Our abstract assertion now becomes the following.

$$\Gamma \vdash p : A \tag{2}$$

stating that p is the proof of A from Γ, where now not only Γ and A are in a formal language, but the proof p is as well.

Aristotle remarked that if someone gave him a putative proof, then he would be able to verify that it is indeed a proof. On the other hand, if someone gave him a putative theorem without a proof, then he would not always be able to verify its correctness. These remarks are the essential intuition behind the twentieth century (metamathematical) propositions that the assertion (2) is decidable (Gödel), while (1) is not (Turing).

A more precise reason why proof checking is decidable is the fact that for the usual systems of type theory one has as a first approximation that

$$\Gamma \vdash p : A \quad \Leftrightarrow \quad \mathrm{type}_\Gamma(p) = A, \tag{3}$$

where type_Γ is a relatively simple computable function. Hence (2) is decidable and one does not need to store the "meta"-proof asserting that $\Gamma \vdash p : A$. We come back to this later, see (14) below.

The decidability of (2) is the essence of why arbitrary mathematical notions, such as a Hilbert space $\mathcal{H}$, can be represented exactly on a computer. In CASs an object like $\sqrt{3}$ can be represented exactly, because we just write it down as a symbol and we know how to manipulate it:

$$(\sqrt{3})^2 + 1 = 4,$$

but $\sqrt{3} + 1$ cannot be rewritten. In PAs arbitrary mathematical notions are similarly written down, but now we also have to store the proofs (or the information that there are proofs) of the more complex properties about them. For example of the property (for $\mathcal{H}$ a topological Hausdorff vector space over $\mathbb{R}$ or $\mathbb{C}$)

$$\mathcal{H} \text{ is locally compact} \quad \Leftrightarrow \quad \mathcal{H} \text{ is finite dimensional.}$$

In the transition from logic to type theory the three aspects that make it possible to express mathematics (many-sortedness, higher-order, constructive disjunction and existence) are incorporated in a natural way. We will now give examples of how mathematical statements can be expressed using higher-order logic and type theory, without having to rely on set theory. For a description of higher-order logic see Schütte (1960); for type theory see Martin-Löf (1984) and Barendregt and Geuvers (2001).

EXPRESSIVENESS

Let $\mathcal{R} = \langle R, +, -, \times, 1, 0\rangle$ be a ring. In Section 1 we defined the syntactic category of expressions over $\mathcal{R}$. We can now define the first-order formulas over this structure.

a	:=	e=e	atomic formulas
f	:=	a | ¬f | f&f | f∨f | f⇒f | ∀vf | ∃vf	(first-order) formulas.

The quantifiers $\forall, \exists$ are intended to range over R. One can state that $\mathcal{R}$ is a commutative ring by the set of the following eight formulas Γ.

$$\begin{array}{rc}
\forall x & x+0=x \\
\forall x,y & x+y=y+x \\
\forall x,y,z & (x+y)+z=x+(y+z) \\
\forall x & x+(-x)=0 \\
\forall x & 1x=x \\
\forall x,y & xy=yx \\
\forall x,y,z & (xy)z=x(yz) \\
\forall x,y,z & x(y+z)=xy+xz.
\end{array}$$

Usually the outer quantifiers $\forall$ are not written. The assertion

$$\Gamma \vdash (x+y)^2 = x^2 + 2xy + y^2$$

states that from Γ one can prove the equation, where of course $e^2 = ee$ and $2e$ abbreviate ee and $e+e$ respectively. Similarly the assertion

$$\Gamma, \forall x,y\,(xy=0 \;\Rightarrow\; (x=0 \vee y=0)) \vdash \forall x\,(4x=0 \;\Rightarrow\; 2x=0),$$

stating that in a commutative ring without zero divisors one has that $4x = 0$ implies $2x = 0$. We have not given the deduction and equality rules that are needed to prove such statements. A small number of these rules will do.

In type theory types serve as different domains for mathematical entities. For example in the theory of vector spaces one needs to distinguish between scalars and vectors; in elementary synthetic projective geometry one has points and lines. Types and their function spaces can also be used to express properties.

A unary property on R can be expressed by introducing a predicate $P_1 : R{\to}\mathtt{Prop}$. A binary predicate has type $P_2 : R{\to}R{\to}\mathtt{Prop}$. The notion of atomic formulas is now extended such that P_1(e) and P_2(e,e) are also included among them.

One can state as follows that a unary $I : R{\to}\mathtt{Prop}$, thought of as set, is an ideal.

$$\begin{array}{rl}\mathtt{ideal}(I) ::= & I(0)\ \&\ \forall x,y{:}R.[I(x)\ \&\ I(y){\to}I(x-y)]\ \& \\ & \forall x,r{:}R.[I(x){\to}I(rx)]\end{array}$$

Properties of binary predicates can be expressed just as easily (instead of $\sim(x,y)$ one writes $x \sim y$).

$$\begin{array}{r}\mathtt{antisymmetric}(\sim) ::= \forall x,y\,[x\sim y \;\to\; y \not\sim x]; \\ \mathtt{irreflexive}(\sim) ::= \forall x\; x \not\sim x.\end{array}$$

In second-order logic there are quantifiers over predicates over R. For example, here is how to express that an element of R has torsion.

$$\mathtt{torsion}(x) ::= \forall P{:}(R{\to}\mathtt{Prop})\,[P(x)\ \&\ \forall y{:}R(P(y){\to}P(y+x)){\to}P(0)].$$

This states that 0 belongs to all sets containing x that are closed under adding x, in particular in $\{x, x+x, x+x+x, \ldots\}$.

Using type theory (with inductive types) this can be expressed more naturally (and one does not even need the second-order quantification).

$$\texttt{Nat} ::= \texttt{zero} \;\texttt{|}\; \texttt{suc Nat} \tag{4}$$

$$\texttt{zero}\, x ::= 0 \tag{5}$$

$$(\texttt{suc}\; n)\, x ::= nx + x \tag{6}$$

$$\texttt{torsion}(x) ::= \exists n{:}\texttt{Nat}.n\, x = 0. \tag{7}$$

Equation (4) defines the type of natural numbers as generated freely from a constant `zero` and a unary operation `suc`(cessor). Equations (5) and (6) define the multiplication of an element of `Nat` with an element of R. Finally, (7) gives the more natural definition of having torsion.

A proper use of second-order quantification (both in logic and type theory) is to express that $\mathcal{R}$ is Noetherian, i.e. every non-increasing (better: weakly descending) sequence of ideals has a fixed point:

$$\begin{aligned}&\forall I{:}(\texttt{Nat}\to(R\to\texttt{Prop})).\\ &\quad[[\forall n{:}\texttt{Nat}.I(n+1) \subseteq I(n)]\to\\ &\quad[\forall n{:}\texttt{Nat}.\texttt{ideal}(I(n))]\to\\ &\quad\exists m.I(m) = I(m+1)].\end{aligned}$$

Here, for $X, Y : R\to\texttt{Prop}$ one defines

$$X \subseteq Y \leftrightarrow \forall r{:}R.X(r)\to Y(r);$$
$$X = Y \Leftrightarrow \forall r{:}R.X(r) \leftrightarrow Y(r).$$

The reason that the notion Noetherian ring is stated in second-order logic is that $\forall I{:}(\texttt{Nat}\to(R\to\texttt{Prop}))$ does not quantify over a type representing ordinary elements, but over a type representing functions. Note that quantifying over $\texttt{Nat}\to(R\to\texttt{Prop})$ essentially is the same as over $\texttt{Nat} \times R\to\texttt{Prop}$ and hence over $\mathcal{P}(\texttt{Nat} \times R)$, the collection of subsets of $\texttt{Nat} \times R$. This is second-order quantification.

Higher-order quantification (both in logic and type theory) can express, for example, that there is a topology on R and that a function $f : R\to R$ is continuous with respect to this topology.

$$\begin{aligned}&\exists\mathcal{O}{:}((R\to\texttt{Prop})\to\texttt{Prop})\\ &\quad[\mathcal{O}(\emptyset)\ \&\ \mathcal{O}(R)\ \&\\ &\quad\forall O_1, O_2{:}(R\to\texttt{Prop})[\mathcal{O}(O_1)\ \&\ \mathcal{O}(O_2)\to\mathcal{O}(O_1 \cap O_2)]\ \&\\ &\quad\forall I{:}\texttt{Set}\forall O{:}(I\to(R\to\texttt{Prop}))[[\forall i{:}I\ \mathcal{O}(O(i))]\to\mathcal{O}(\textstyle\bigcup_{i:I} O(i))]\\ &\quad\&\\ &\quad\forall O{:}R\to\texttt{Prop}\ \mathcal{O}(O)\to\mathcal{O}(f^{-1}(O))].\end{aligned}$$

Here, one has $\emptyset, R, \bigcup_{i:I} O(i), f^{-1}(O) : R\to\texttt{Prop}$ defined as usual:

$$\begin{aligned}\emptyset(x) &::= \texttt{False};\\ R(x) &::= \texttt{True};\\ (\textstyle\bigcup_{i:I}O(i))(x) &::= \exists i{:}I\ O(i)(x);\\ (f^{-1}(O))(x) &::= O(f(x)).\end{aligned}$$

The quantification over $((R{\to}\texttt{Prop}){\to}\texttt{Prop})$ is really the same as over $\mathcal{P}(\mathcal{P}(R))$, which is third order.

In this way most mathematical statements can be formulated smoothly in higher-order logic and type theory. (Statements in finite combinatorics are sometimes hard to formulate—even more so to prove—in type theory, because they rely on intuition. Take for example the combinatorial theorem: *It is impossible to give a disjoint subdivision of a cube into finitely many smaller ones of unequal size.* This result has a proof with a simple geometric intuition which is hard to catch formally. See also Jamnik (1997).)

DEDUCTIVE POWER

In the usual logical systems deductive power comes in conjunction with expressive power. This means that if there are first- (or higher)- order quantifiers in the language, then there are also deduction rules that govern these. This means that then one is able to prove the "valid" sentences involving these quantifiers. In principle matters can be separated: adding quantifiers as a means of expression and adding the quantifier rules as a means of assertion. Apart from some technical matters concerning substitution the rules for quantification are not hard. For example, for universal quantification the rules are as follows.

$$\frac{\Gamma \vdash \forall x{:}D.Ax \quad \Gamma \vdash d : D}{\Gamma \vdash Ad} \qquad \frac{\Gamma, x{:}D \vdash Ax}{\Gamma \vdash \forall x{:}D.Ax}.$$

Here, D is to be interpreted as a set and $d : D$ as $d \in D$. The first rule was already used by Aristotle.

In type theory, where proofs become explicit, these rules become as follows.

$$\frac{\Gamma \vdash p : \forall x{:}D.Ax \quad \Gamma \vdash d : D}{\Gamma \vdash pd : Ad} \qquad \frac{\Gamma, x{:}D \vdash q : Ax}{\Gamma \vdash \lambda x{:}D.q : \forall x{:}D.Ax}.$$

The intuition is that p is a proof of the universal statement $\forall x{:}D.Ax$ because it is a function such that for all $d \in D$ the value $pd (= p(d)$ in more traditional notation) is a proof of Ad. The expression $\lambda x{:}D.q$ denotes the function that assigns to $x \in D$ the value q (that may depend on x). For a precise description of some type theoretic systems used in type checking, see Barendregt and Geuvers (2001). This way of handling proofs and their rules causes the logic formalized by type theory to be intuitionistic. We will make use of this in Section 5.

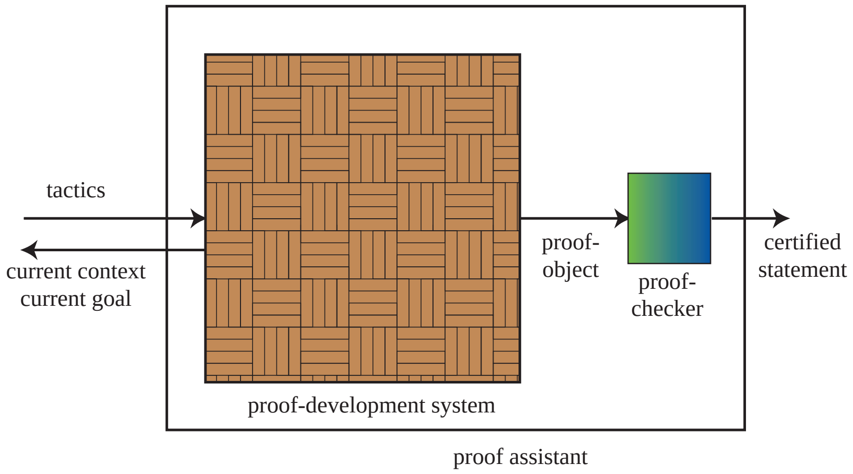
Type theory as a medium to formalize proof for proof checking was initiated by de Bruijn in the 1960s, see Nederpelt *et al.* (1994) for an overview. In Martin-Löf (1984) emphasis has been put on special kinds of type theory with a predicative (read: reliable) semantics.

4. Proof Assistants

A proof assistant is a computer program that allows a user to formulate and input an axiomatic theory as well as definitions and statements in the language of that theory. The assistant then checks whether the input is mathematically well formed. In addition, the user can propose a statement as the goal (to be proved) and the assistant then helps to construct a proof. The assistance offered for finding a proof ranges from none (in a pure proof checker) to everything (in a theorem prover, only possible for restricted mathematical areas such as Euclidean geometry), with in between an interactive proof development system.

RELIABILITY

One may wonder whether a proof constructed partially or totally by a computer is reliable. Indeed, software is complex and hence it is hard to avoid errors. There is, however, a satisfactory methodology that warrants correctness of statements to the highest degree of certainty. A proof should be in such a format that in order to check it, one needs only to check whether each step is correct according to a small number of axioms and deduction rules. The proof may be large, so that the checking takes time, but the software that performs the checking consists of a small program. This program then has to be verified "by hand" and one obtains a reliable system, even if the proof development part of the assistant is large (and hence error prone). As a result one has proof assistants of the following general architecture producing statements with the highest degree of certainty. Such statements are said to be "evidently impeccable", the evidence coming from the proof object.



In the picture the notion "proof object" refers to the fully formalized proof that is either stored or ephemeral and can be used in order to ensure that the checking is reliable. The expression "tactics" refers to the commands that the user has to give to help build the proof in an interactive way.

A proof assistant satisfies the *de Bruijn criterion* if it has a proof checker that is small enough to be verified by hand. Proof assistants that have proof objects that are stored have the advantage of the possibility of *independent checking*. This means that the proof object can be sent to another site where it is checked by a locally trusted proof checker. Additionally, proof objects are useful, because from them one may extract certified algorithms in an automated way. These extracted algorithms can be surprisingly efficient, in fact more efficient than the original handmade ones.

COMPUTATIONS AND PROOFS

Computations are frequently used in mathematics in order to obtain answers to numerical problems. Proofs have been used in order to warrant properties. An important spin-off is the possibility of making substantial shortcuts in computations while being sure that the same outcome is obtained.

There are also symbolic computations. These help by making shortcuts not only in other computations (e.g. numerical ones) but also in proofs. We give some short examples of computations useful in proofs.

(1) Let $A_n = (..((\text{true}\rightarrow \underbrace{q)\rightarrow q)\ldots\rightarrow q}_{n\times q})$, where q is some propositional variable. In order to define A_n, write

$$A_0 = \text{true};$$
$$A_{n+1} = A_n \rightarrow q.$$

The goal is to show $\vdash A_{2000}$. This can be done by first proving $\forall k\ [(A_{2k} \leftrightarrow \text{true})\ \&\ (A_{2k+1} \leftrightarrow q)]$. Then the goal follows from the computation that $2000 = 2 * 1000$.

(2) In order to show that in a ring one has

$$\forall x\ (x^3 + x^2 + x + 1)(x-1) = x^4 - 1,$$

a simple but essential symbolic computation is needed.

(3) For an $n \times n$ matrix A over $\mathbb{Q}$ one has

$$A \text{ is invertible} \quad \Leftrightarrow \quad \det(A) \neq 0.$$

This is a clear transformation of a property into a computation.

Since computing is important for proving, one would like that if f is a computable function (on a freely generated algebra $\mathcal{A}$), then there is a formal expression $F(x)$ such that for all $a, b \in \mathcal{A}$

$$f(a) = b \quad \Leftrightarrow \quad \vdash F(\underline{a}) = \underline{b}, \tag{8}$$

for some representation $a \mapsto \underline{a}$ of elements of $\mathcal{A}$ in the theory.

EFFICIENCY

The most efficient way (from the point of proving) to ensure (8) is to add for each computable function f an expression $F(x)$ and postulate axiomatically that for arbitrary $a \in \mathcal{A}$

$$\vdash F(\underline{a}) = \underline{f(a)}. \tag{9}$$

This is what is called in Barendregt (1997) the Poincaré Principle. In Poincaré (1902) it is argued that if $2+2=4$ is needed in a mathematical argument, then this part is not so much a proof in the strict sense of the word, but a “mere verification”. Adding (9) as an axiom is possible, even if we want proof checking to be decidable. (For this reason it was required that the function f be computable.) A particular way to ensure that (9) is automatically derivable is to take as deduction rule (in type theory)

$$\frac{\Gamma \vdash p : A(F(\underline{a}))}{\Gamma \vdash p : A(\underline{f(a)})} \tag{10}$$

(a similar rule for logic is obtained by leaving out “p :” twice). In type theory with proof objects this is quite efficient, since the p does not need to be changed in order to acknowledge the equality. Equation (9) is then derived as follows.

$$\frac{\Gamma \vdash \text{refl} : F(\underline{a}) = F(\underline{a})}{\Gamma \vdash \text{refl} : F(\underline{a}) = \underline{f(a)}},$$

by taking $A(x) \equiv (F(\underline{a}) = x)$ in (10). Here, "refl" is a proof of the reflexivity of equality. A useful way in which (10) is incorporated in modern PAs is the following.

$$\frac{\Gamma \vdash p : A(t)}{\Gamma \vdash p : A(t')} \quad t =_R t', \tag{11}$$

where the equality $t =_R t'$ is given by some term rewriting system (TRS) R, see Klop (1992), and can be decided externally. In this way one has, for example, that

$$\frac{\Gamma \vdash p : A((x^3 + x^2 + x + 1)(x - 1))}{\Gamma \vdash p : A(x^4 - 1)}$$

by employing (11) for the rewriting relation "expand" for ring expressions.

If (11) is added to type theory, then type (and hence proof) checking becomes

$$\Gamma \vdash p : A \Leftrightarrow \text{ type}_\Gamma(p) =_R A. \tag{12}$$

But then the de Bruijn criterion (having a simple proof checker that can be verified by hand) is seriously impaired, because verifying whether $=_R$ holds between two expressions may need a complex algorithm. The same is true if one of the other forms of the Poincaré Principle (8), (9), (10) or (11) is added as a deduction rule. Therefore these forms of the Poincaré Principle are in conflict with the de Bruijn criterion. Another disadvantage of rule (11) is that we would also like to use term rewriting for operations that do not preserve equality. For example

$$x^3 \to_{\text{dif}} 3x^2$$

is a way of implementing derivation in CASs; such an algorithm we would like to have as well in PAs.

Both problems have a methodologically elegant solution by considering (11) only for some specific rewrite relations β, δ, ι:

$$\frac{\Gamma \vdash p : A(t)}{\Gamma \vdash p : A(t')} \quad t =_{\beta\delta\iota} t'. \tag{13}$$

Here, $=_\beta$ is the rewrite rule of the lambda calculus:

$$(\lambda x.t)s \to_\beta t[x := s],$$

where $[x := s]$ denotes substitution of the expression s for the (proper) occurrences of the variable x. These rewrite systems allow arbitrary manipulations with formulas.

The rewrite relation δ is related to definitions. Without these, statements become too complex to be formulated and to be proved. Definitions are abbreviations, to be unfolded, whenever needed, to what they stand for. The rewrite equality $=_\delta$ is used to expand definitions, for example

$$\cosh x \to_\delta \frac{e^x + e^{-x}}{2}.$$

Finally, the rewrite equality $=_\iota$ is used to implement primitive recursion:

$$\begin{aligned} \text{fac}(0) &\to_\iota 1 \\ \text{fac}(x+1) &\to_\iota (x+1) * \text{fac}(x) \end{aligned}$$

for the factorial function $\mathrm{fac}(x) = x!$ or

$$\begin{aligned}\mathrm{mirror}(\bullet) &\to_\iota \bullet\\ \mathrm{mirror}([t,s]) &\to_\iota [\mathrm{mirror}(s), \mathrm{mirror}(t)]\end{aligned}$$

for the function that mirrors binary trees with $\bullet$'s at the leaves.

Having the tools of $\beta\delta\iota$-reductions makes it possible to replace rewrite operations such as

$$x^3 \to_{\mathrm{dif}} 3x^2$$

with

$$F_{\mathrm{dif}}(x^3) \to_{\beta\delta\iota} 3x^2,$$

thereby replacing the "special purpose machine" $\to_{\mathrm{dif}}$ by the more "universal machine"[†] $\to_{\beta\delta\iota}$ and "software" F_{dif}.

One advantage of this transition is that in order to ensure the reliability of the proof checker, one only has to verify the implementation of $=_{\beta\delta\iota}$ once, thereby not diverging much from the original de Bruijn criterion. Indeed, proof and type checking become

$$\Gamma \vdash p : A \Leftrightarrow \ \mathrm{type}_\Gamma(p) =_{\beta\delta\iota} A. \tag{14}$$

The second "advantage"[‡] is that now (functional) programs, e.g. for integration and simplification, have to be explicitly provided. Once we have these programs in our formal language, one has a handle to state and prove required properties. In Jackson (1995) one can find many uses of this methodology. In Armando *et al.* (1999), some support is given towards the synthesis of such programs.

A different approach is taken in the PAs HOL, see Harrison (1996), and *Isabelle* (2001). In these systems one does not adopt the Poincaré Principle, but produces "non-standard" proof objects. These contain all information to be unfolded to a complete proof object (including traces of computation); because these are rather large, this process happens *over time.* The advantage is increased space efficiency; the disadvantage is that there are no proof objects at one moment in time that can be manipulated (for example to be modified or to extract algorithms from them).

5. Forms of Communication

The communication between CASs and PAs is somewhat asymmetric. CASs have an essentially less rich (although at present more developed) language than PAs. CASs deal with mathematical objects and (computable) ways to manipulate these. PAs on the other hand also deal next to these with (possibly) non-computable properties of these objects. And the richer PAs based on type theories also incorporate the manipulations on objects and their properties (specifications) as standard objects.

[†]The expression "universal machine" was put in quotes because $=_{\beta\delta\iota}$ does not provide a computation model for all computable functions. Wanting to improve on this one can add Y-reduction for fixed-point recursion. If this is done proof checking is no longer decidable, but semi-decidable: sometimes one has to possibly wait an infinitely long time. This is the price one pays for universality. But if the verification process halts, then one is sure to have reached a proof that can also be reached without the Y-reduction and hence is reliable, see Geuvers *et al.* (1999).

[‡]This advantage is real, but laborious: one has to give correctness proofs for many algorithms. Therefore in the industrially used proof assistant PVS one essentially uses deduction rule (10) for a considerable library of useful functions. Although as a result the system PVS itself is not impeccable, its use makes it possible to obtain a substantial reduction of bugs in industrial software.

Suppose that in a given mathematical situation we want to prove a goal for which some computation is needed. We can represent this as the following query in logic

$$\Gamma \vdash A(\underline{a}, ?) \tag{15}$$

and in type theory

$$\Gamma \vdash ?? : A(\underline{a}, ?). \tag{16}$$

kIndeed, we not only want the data b on a data domain such that $\Gamma \vdash A(\underline{a}, \underline{b})$, but also a proof of this fact. A typical example of such a situation is that there is a function f on D such that for all $a, b \in D$ one has

$$f(a) = b \Leftrightarrow \Gamma \vdash A(\underline{a}, \underline{b}).$$

This is a situation in which PAs and CASs can fruitfully collaborate. There are several ways in which this can be done. We describe these from the point of view of the PAs.

AUTARKIC

At this level, the PA acts as an independent "autarkic" entity: it finds the b by an internal function F and the proof by an internal proof generator pg:

$$\Gamma \vdash \mathrm{pg}\ \underline{a} : A(\underline{a}, F\underline{a}).$$

This is a reasonable situation: the value b and the required correctness proof can be found in a reliable way. However, the construction of F and pg may be hard. Moreover, the evaluation of $F\underline{a}$ to $\underline{b}$ may be of a high complexity. Since PAs are (at the current state of the art) no champions in evaluation, one may want to follow a different strategy.

But before doing so, we discuss one of the more spectacular examples in terms of formalized computer algebra: *the Buchberger algorithm.* It is implemented in most standard general purpose CASs (e.g. Maple, Mathematica, MuPAD, Reduce, Singular). When given a finite set B of elements of a polynomial ring $\mathcal{R}$ over a field (allowing for computable arithmetic as described in Section 2), it produces a finite set P of generators of the ideal $B\mathcal{R}$ generated by B with the property that, by means of a simple rewrite procedure in $\mathcal{R}$, for each $a \in \mathcal{R}$, a unique representative of the class $a + B\mathcal{R}$ containing a can be computed. The finite set P is usually called a *Gröbner basis* of $B\mathcal{R}$, its computation can be carried out by the Buchberger algorithm. The Gröbner basis depends on the choice of an ordering on the monomials in $\mathcal{R}$ satisfying certain technical properties; such orderings are referred to as a *reduction ordering.* Although the Buchberger algorithm has been around since 1965, it has only recently been established by an autarkic PA: in Théry (1998) a Buchberger algorithm in a PA together with a correctness proof is constructed.

Let us consider a specific instance.

Claim. *Suppose we are given the ring $\mathcal{R} = \mathbb{Q}[x, y]$. Define the lexicographic reduction ordering on $\mathcal{R}$ with $x > y$. Consider the ideal $I = \{x^2y - 1, xy^2 - 1\}\mathcal{R}$. Then $P := \{x - y, y^3 - 1\}$ is a Gröbner basis of I.*

Here we have an example of the technological use of intuitionistic logic. In classical mathematics one can state the following.

Theorem. *For each ideal I in $\mathcal{R}$ there is a Gröbner basis P of I.*

The fact that P depends on I in a computable way cannot be expressed very well in the traditional Bourbaki style of formalizing mathematics in set theory. One can only state this via a formalization of Turing machines, but this is very awkward. For the computable dependency it is not sufficient to state:

> *There is a Turing machine TM such that for every ideal I in $\mathcal{R}$ generated by $r_1, \ldots, r_n \in \mathcal{R}$ the result TM$(\{r_1, \ldots, r_n\}) = P$ is a Gröbner basis of I.*

This is because using classical logic the existence claim "There exists" does not need to have a witness! The only way to state that there is a computable dependency is to actually fully describe the Turing machine TM and state:

> *For each ideal I in $\mathcal{R}$ generated by $r_1, \ldots, r_n \in \mathcal{R}$ the result TM$(\{r_1, \ldots, r_n\}) =$ P is a Gröbner basis of I.*

It is completely unnatural to require the description of a Turing machine (or even a program in a higher level computational model) in the formulation of a mathematical statement. In formalizations using type theory, with its intuitionistic logic statement (5) implies that one has an algorithm. (And the proof in Théry (1998) does in fact have the obligation to exhibit a form of the Buchberger algorithm.) If one wants to state (5) without claiming that the dependency is computable, then one should replace $\exists P$ by the weak existence discussed before.

Theorem. $\forall I \subseteq \mathcal{R}[I$ *is an ideal* $\Rightarrow \neg\forall P \neg$ $[P$ *is a Gröbner basis for* $I]]$.

For example the classical statement,

Theorem. *Every principal ideal domain is a unique factorization domain*

has to be formalized in this way.

skeptical

Now the PA accepts the superiority of the CAS and asks for a given a the value of b. But the PA does this in a skeptical way, it wants to have a warranty. There are several ways in which this can be done. The first approach gives the PA some self esteem: besides the output b the CAS also produces a witness c $(= c_a)$ that is such that the PA can employ a proof generator pg using this witness:

$$\Gamma \vdash \text{pg}\ \underline{a}\ \ \underline{c_a} : A(\underline{a}, \underline{b_a}).$$

Of course the b_a and c_a depend on the a, but that is the responsibility of the CAS, the PA has no power here. By way of an example, consider the following assertion.

Claim. $\gcd(14, 30) = 2$.

Here we use the following lemma, which can be formally proved.

Lemma. *For all $m, n, d \in \mathbb{N}$ we have*

$$\gcd(m, n) = d \Leftrightarrow \exists x, y \in \mathbb{Z}\ [xm + yn = d\ \&\ d\,|m\ \&\ d\,|n].$$

When a skeptical PA presents a CAS with the query "gcd(22,30) = ?", then the CAS might answer "2, witness $x = -4, y = 3$". This is done using the so-called "extended gcd" algorithm. (For the Gröbner basis algorithms there are similar extended versions.) The PA then checks $-4 \cdot 22 + 3 \cdot 30 = 2,\ 2|22$ and $2|30$ and the lemma can be applied. (For the divisibility some help in the form of a witness may also be given.)

There are also algorithms that do not need to produce witnesses at all (next to the output). Classical examples are factorization of integers and indefinite integration of elementary functions; these are based on the availability of "proven" algorithms for the verification, namely multiplication of integers and differentiation of elementary functions.

An experiment has been carried out, providing full proofs of the primality of (relatively) large primes using the Pocklington criterion (based on Fermat's little theorem). The CASs provide witnesses; among these there is a sequence of smaller primes to which the method is applied recursively. In this way the proof assistant Coq and the computer algebra system GAP have collaborated and fully verified that

1111111111111111111	is prime;
9026258083384996860449366072142307801963	is prime.

See Caprotti and Oostdijk (2001) for details. The primes can be found on the web page of Honaker (2000). The correctness of these statements cannot be proved by Coq or found out to hold by GAP alone.

There is a variant of this method in which the witnesses are bigger. It may be the case that the witness is not a simple value in D but a bigger expression C_a. Then one has the situation

$$\Gamma \vdash \text{pg}\ \underline{a}\ \ C_a : A(\underline{a}, \underline{b_a}).$$

The PA still has the responsibility to do some computation.

Problem. Given two square matrices A, B over $\overline{\mathbb{Q}}^n$, determine whether they are conjugate.

When presented with two matrices A and B, the CAS may answer "Yes, via the invertible matrix C." or "No, since there are invertible matrices C, D such that CAC^{-1} and DBD^{-1} have different Jordan blocks." The PA will then be able to complete its homework, which involves the verification that $B = CAC^{-1}$ in the former case and that CAC^{-1} and DBD^{-1} are both in Jordan normal form but are distinct matrices in the second case.

At the other end of the scale the CAS comes up with both the output b and the proof p_a:

$$\Gamma \vdash p_a : A(\underline{a}, \underline{b_a}).$$

An example of this "worse" case skeptical interaction is Todd–Coxeter coset enumeration. It is implemented in the two CASs GAP and MAGMA which are specially equipped for group theory. See Cohen and Sterk (1999) for an introduction to the algorithm, which is really a semi-algorithm in the sense that it does not need to terminate. Given a group G in terms of generators and relations, and a subgroup H of G in terms of generators, which are words in the generators of G, it constructs the cosets Hg with $g \in G$, *if* the number of cosets is finite.

Define

$$\begin{aligned} G := {} & \text{free group on } a,b,c,d \text{ modulo the relators} \\ & a^2, b^2, c^2, d^2, (ab)^3, (dc)^3, (ca)^3, (cb)^4, \\ & (cbab)^4, (abc)^7, (ad)^2, (bd)^2. \\ H := {} & \langle a,b,c \rangle. \end{aligned}$$

The algorithm will list H, Hd, $Hda = Had = Hd, \ldots$. On the above input, the computer algebra program GAP finds exactly 15 360 cosets. All identifications of cosets can be certified by relations found during the coset enumeration. Some analysis of the output shows that the group G has a normal subgroup N of order 2^7 with quotient isomorphic to the alternating group on eight letters. But the only way to guarantee that there is no bigger group; that is, that there are indeed no more than 15 360 cosets, is to "trace" the algorithm: to feed each step of the execution of the algorithm into a proof. Clearly, this is a most undesirable situation. We have come to the other extreme of the so-called skeptic approach: in principle (and sometimes never in practice), there is a way of supplying credence to the answer.

As we will see, there are even cases in which complexity of proof checking forces the PA to be an obedient pupil in the traditional sense.

BELIEVING

From time to time a PA just has to believe a result from a CAS. This can be represented by

$$\Gamma \vdash \text{axiom} : A(\underline{a}, \underline{b_a}),$$

stating that $A(\underline{a}, \underline{b_a})$ is believed axiomatically. In science and engineering such a situation often happens, for mathematics it is relatively new.

An example of a situation in which mathematical software cannot come up with anything formal but the answer is the following.

PROBLEM. Does there exist a projective plane of order 10?

Assisted by a special purpose CAS one is given the answer "No". An exhaustive computer search has not found a projective plane of order 10, see Lam (1991). For the sake of argument, we refer to the intelligent software used here as a special purpose CAS. Since the search required all available machine power, no trace could be left as a witness to the conclusion that there is no projective plane of order 10. As a matter of fact, this observation does not do real credit to the authors. They have given lots of indications of how the search is conducted, how internal cross checks were run, and which configurations were considered.

A different category of statements for which one needs to believe is related to the algorithms that establish up to a probability of almost 100% that something holds, e.g. that a number p is prime, see Ribenboim (2000). Actually such statements can be quite reliable. In any case they can be used in combination with other statements for which there is no proof at all and one relies on the authority of a CAS.

6. Conclusions

In principle, PAs have the potential to become autarkic tools to help a human develop mathematical theories in an evidently impeccable way. Because CASs are[†] more efficient in computing, collaboration between PAs and CASs is profitable. The PAs are good for reliability and the CASs for efficiency. Several forms of collaboration are possible with varying trade-offs between reliability vs. efficiency.

For the communication one will need to make use of several techniques.

(1) A common language. The emerging standard *OpenMath*, see Caprotti and Cohen (2001), can be used to carry statements of CASs and PAs. A good language to produce OpenMath documents with semantically rich mathematical objects is OMDoc described in Kohlhase (2000a,b).
(2) Adding the possibility of making standardized queries to the PAs. For example $\Gamma \vdash ?? : A?$.
(3) Agent technology in order to communicate queries and answers; these are the web counterparts for input and output between PAs and CASs, see Armando *et al.* (2000).

Using PAs in the autarkic or skeptical way produces evidently impeccable results. It may be the case that statements obtained by theorem provers not satisfying the de Bruijn principle are also impeccable. But probably the most cost effective way to obtain impeccability is to take care that there is evidence, i.e. to rely on the methodology of proof checking.

For the information given by a CASs it would be useful to have, next to the usual complexity analysis of mathematical algorithms, a convincibility analysis. The goal of this analysis is to express to what extent a CAS, when communicating across the Web (or with a PA), is capable of providing evidence that the answer given to a query is impeccable.

Acknowledgements

Use has been made of the package “prooftree” of Paul Taylor to draw derivations. We thank Olga Caprotti and Martijn Oostdijk for providing us with large examples of formally proved prime numbers, the referees for their comments and Don Taylor and Weldon Lodwick for suggesting some language improvements.

References

Adams, W. W., Loustaunau, P. (1994). *An Introduction to Gröbner Bases.* AMS, Oxford University Press.

Armando, A., Smail, A., Green, I. (1999). Automatic synthesis of recursive programs: the proof-planning paradigm. *Automated Software Engineering*, **6**, 329–356.

Armando, A., Kohlhase, M., Ranise, S. (2000). Communication protocols for mathematical services based on KQML and OMRS, *Eighth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (CALCULEMUS-2000).*

Barendregt, H. (1997). The impact of the lambda calculus in logic and computer science. *Bull. Symb. Logic*, **3**, 181–215.

[†]At present, but probably always for PAs with proof objects; for the PAs with proof traces spread out over time such as HOL and Isabelle it may be possible to become fully autarkic in an efficient way.

Barendregt, H., Barendsen, E. (2001). Autarkic computations in formal proofs. *J. Symb. Comput.*, to appear.

Barendregt, H., Geuvers, H. (2001). Proof checking using dependent type systems. In Voronkov, A., Robinson, A. eds, *Handbook of Automated Reasoning.* Amsterdam, Elsevier, to appear.

Buchberger, B. (1985). Recent trends in multidimensional system theory. In Bose, N. K. ed., *Reidel, Chapter Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory*, pp. 184–232.

Buchberger, B., Winkler, F. eds (1998). *Gröbner Bases and Applications*, Cambridge University Press.

Caprotti, O., Cohen, A. (2001). On the role of OpenMath in interactive mathematical documents. *J. Symb. Comput.*, Special Issue on the Integration of Computer Algebra and Deduction Systems, to appear.

Caprotti, O., Oostdijk, M. (2001). Formal and efficient primality proofs by use of computer algebra oracles. *J. Symb. Comput.*, submitted.

Cohen, A. M. (2000). Communicating mathematics across the web. In Engquist, B., Schmid, W. eds, *Mathematics Unlimited—2001 and beyond*, pp. 283–300. Springer.

Cohen, A. M., Cuypers, H., Sterk, H. eds (1999). *Some Tapas of Computer Algebra.* Springer.

Cox, D., Little, J., O'Shea, D. (1992). *Ideals, Varieties, and Algorithms.* Springer.

Eisenbud, D. (1995). *Commutative Algebra with a View Toward Algebraic Geometry.* Springer.

Geuvers, H., Poll, E., Zwanenburg, J. (1999). Safe proof checking in type theory with Y. In Flum, J., Rodriguez-Artalejo, M. eds, *Computer Science Logic (CSL'99)*, LNCS **1683**, pp. 439–452. Springer.

Harrison, J. (1996). Proof style. In Giménex, E., Paulin-Mohring, C. eds, *Types for Proofs and Programs: International Workshop TYPES'96*, *LNCS* **1512**, pp. 154–172. Aussois, France, Springer-Verlag.

Honaker, G. L. (2000). Prime curios!, Available at `URL:<http://www.utm.edu/research/primes/curios/index.htm>`.

Isabelle (2001). Proof Assistant. Homepage `URL:<isabelle.in.tum.de>`.

Jackson, P. (1995). Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra. Dissertation, Cornell University.

Jamnik, M. (1997). Automation of diagrammatic proofs in mathematics. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, p. 1541. San Francisco, Morgan Kaufmann Publishers.

Klop, J. W. (1992). *Term Rewriting Systems, Handbook of Logic in Computer Science*, volume 2, pp. 1–116. New York, Oxford University Press.

Kohlhase, M. (2000a). OMDoc: An infrastructure for OPENMATH content dictionary information. *Bulletin ACM Special Interest Group Symb. Automat. Math (SIGSAM)*, **34**, 43–48.

Kohlhase, M. (2000b). OMDoc*: Towards an* OPENMATH representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes. Available at `URL:<http://www.mathweb.org/omdoc>`.

Lam, C. W. H. (1991). The search for a finite projective plane of order 10. *Am. Math. Monthly*, **98**, 305–318.

Martin-Löf, P. (1984). *Intuitionistic Type Theory, Studies in Proof Theory.* Bibliopolis, Napoli.

Nederpelt, R. P., Geuvers, J. H., de Vrijer, R. eds (1994). *Selected Papers on Automath.* Amsterdam, North-Holland Publishing Co.

Poincaré, H. (1902). *La Science et l'Hypothèse*. Paris, Flamarion.

Ribenboim, Paulo (2000). *My Numbers, My Friends, Popular Lectures on Number Theory.* New York, Springer-Verlag.

Schütte, Kurt (1960). Syntactical and semantical properties of simple type theory. *J. Symb. Logic*, **25**, 305–326.

Théry, Laurent (1998). A certified version of Buchberger's algorithm. In *Automated deduction—CADE-15 (Lindau, 1998)*, pp. 349–364. Berlin, Springer.

Originally Received 1 November 2000
Accepted 15 February 2001