On the Design of Mathematical Concepts*

Manfred Kerber[i] Martin Pollet[i,ii]

[i]School of Computer Science
The University of Birmingham, Birmingham B15 2TT, England
{M.Kerber|M.Pollet}@cs.bham.ac.uk, http://www.cs.bham.ac.uk/

[ii]Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany
pollet@ags.uni-sb.de, http://www.ags.uni-sb.de/

November 2003

Abstract

That foundational systems like first-order logic or set theory can be used to construct large parts of existing mathematics and formal reasoning is one of the deep mathematical insights. Unfortunately it has been used in the field of automated theorem proving as an argument to disregard the need for a diverse variety of representations. While design issues play a major rôle in the formation of mathematical concepts, the theorem proving community has largely neglected them. In this paper we argue that this leads not only to problems at the human computer interaction end, but that it causes severe problems at the core of the systems, namely at their representation and reasoning capabilities.

> *It would be somewhat misleading to infer that foundational systems act primarily as a basis out of which mathematics is actually created. The artificiality of that view is evident when one reflects that the essential content of mathematics is already there before the basis is made explicit, and does not depend on its existence.*
>
> R. Goldblatt *[Gol84]*

1 Introduction

Automated deduction tools are built with a wide range of expectations in mind. They may serve as educational systems, be used in the study of formal systems, as proof checkers (with different levels of sophistication) and automated theorem

*This work was supported by European Commission IHP Calculemus Project grant HPRN-CT-2000-00102.

provers. In spite of partly impressive reasoning power of such systems, in many standard applications the systems are harder to use than one would hope.

Let us take a closer look at proof checking, for instance. This task should be easy to do (since checking a proof is supposed to be easy compared to finding it), but in many application areas it is not easy at all to perform this task. The deterrent to use a computer-based system is often so high that typically they are not used – even by the developers of the systems for their own proofs. Why is that so? Is it because their interfaces are too poor? Or is there a more fundamental problem with them? Surely improving interfaces can strongly help to improve existing systems, but we will argue that the main problem is more fundamental. Standard systems are rigid in that they disregard many important design issues which go beyond interface issues, but are concerned with the reasoning possibilities themselves.

In the traditional theorem proving community, which we consider as our home community, a seemingly strong argument against new approaches in theorem proving has been of the type "Everything you can do in your system X, I can do in Y," where Y stands typically for standard first-order logic. This kind of thought is so strong that proponents of the conventional approach to theorem proving seem to fail to even understand why this argument – albeit true – may be unhelpful and misleading.

For instance, Pat Hayes said in 1974 ([Hay74] quoted from [BL85, p.18]):

> *A more recent attack on conventional theorem-proving ... is that it is too concerned with "machine-oriented" logic, and not enough with "human oriented" logic. I confess to being quite unable to understand what this could possibly mean.*

In this paper we try to clarify why the argument, although it may be technically correct, is pragmatically flawed. The argument is pragmatically wrong, since it is meant to say "Your system X is redundant and uninteresting, since we have system Y already, which suffices for everything you possibly may want to do." Since this argument was widely accepted the focus in the field was set much too narrow on the study of foundational systems, while vital aspects were eliminated from investigation.

Of course there are exceptions and we claim in no way that we are the first to have a look at this relationship of mathematical practice and fundamental systems. In this paragraph we do not claim to give a comprehensive overview of this type of work. We just mention some work in this direction, which we think provides very important starting points to the support of the design of mathematical concepts. In Automath, N.G. de Bruijn developed the idea of a mathematical vernacular [Bru94], which should allow to write everything mathematicians do in informal reasoning, in a computer assisted system as well. In this tradition, Hugo Elbers looked in [Elb98] at aspects of connecting informal and formal reasoning, in particular the integration of computations in formal proofs. Francis Jeffry Pelletier [Pel91] as well as Henk Barendregt and Arjeh Cohen [BC01] discuss related philosophical questions on the nature of proof. Proof planning [Bun88] in general can be viewed as an attempt to simulate informal reasoning (and integrate it with formal reasoning). Following this

paradigm, Alan Bundy made first steps towards a Science of Reasoning [Bun91], which goes beyond a narrow focus on a particular calculus. Ursula Martin took in [Mar99] a close look at the mathematical practice and its relationship to computer algebra and computer-assisted reasoning. Michael Beeson presented in [Bee98] a system that combines deductive and computational reasoning steps. He proposed to use the mathematical standard for checking the correctness of proofs generated by the system, namely peer reviews.

We are not aware, however, of work in which the design process in mathematics is compared to the design possibilities of computer based mathematical support systems. In this paper, we look at design problems, which current automated reasoning systems suffer from, and relate them in the rest of this introduction to an old debate in mathematical philosophy, which summarises the relationship between foundational systems and informal systems very well.

We do not doubt that one of the deepest insights in the foundations of mathematics resulted from the epochal work by Bertrand Russell and Alfred North Whitehead. Russell articulated the idea in the *Principles of Mathematics* [Rus03] namely, to reduce mathematics to formal logic, and carried it through together with Whitehead in the famous *Principia Mathematica* [WR10] – a work often quoted and seldom read. Russell was at this time also in inspiring discussions with Ludwig Wittgenstein and strongly acknowledges Wittgenstein's contribution to his thoughts. He writes in [Rus18] (quoted from the reprint in [Rus56, p.178]):

> *As I have attempted to prove in The Principles of Mathematics, when we analyse mathematics we bring it all back to logic. It all comes back to logic in the strictest and most formal sense.*

Although there is evidence that Wittgenstein shared Russell's view, he later took the opposite stance and attacked Russell's approach. In particular he discusses the important notion of proof (quoted from [Wit56, p. 143]):

> *'A mathematical proof must be perspicuous.' ... I want to say: if you have a proof-pattern that cannot be taken in, and by a change in notation you turn it into one that can, then you are producing a proof, where there was none before.*

One of the reasons why the Principia are so rarely read is that the main ideas of the proofs are no longer visible in very long and very detailed proofs. Wittgenstein continues (p. 176f) to question the idea to try to reduce everything to a very small number of primitives (had the resolution calculus already been invented at that time his attack might have been to try to reduce everything to one single rule):

> *Mathematics is a* MOTLEY *of techniques of proof. – And upon this is based its manifold applicability and its importance. ...*
> *Now it is possible to imagine some – or all – of the proof systems of present-day mathematics as having been co-ordinated in such a way with one system, say that of Russell. So that all proofs could be carried out in this system, even though in a roundabout way. So*

> *would there then be only the single system – no longer the many? – But then it must surely be possible to shew of the one system that it can be resolved into the many. – One part of the system will possess the properties of trigonometry, another those of algebra, and so on. Thus one can say that different techniques are used in these parts.*

He continues then to counter the argument that Russell and Whitehead have constructively shown the possibility to reduce everything to one single system (p. 185)[1]:

> *If someone tries to shew that mathematics is not logic, what is he trying to shew? He is surely trying to say something like: – If tables, chairs, cupboards, etc. are swathed in enough paper, certainly they will look spherical in the end.*
> *He is not trying to shew that it is impossible that, for every mathematical proof, a Russellian proof can be constructed which (somehow) 'corresponds' to it, but rather that the acceptance of such a correspondence does not lean on logic.*

We try to exemplify in the rest of the paper why these matters are crucial for automated theorem proving systems. One important issue is that of acceptance among mathematicians. Current automated theorem provers do not find acceptance among mathematicians, while computer algebra systems do. On first sight this is surprising since proving theorems can be considered as much more a main activity of mathematicians than calculating and computing. We believe that it has to do with the fact that in many computer algebra systems things are *as they should be, as an inexperienced but mathematically educated user would expect them to be*. That is, these systems are typically *well-designed*. In automated theorem proving systems they are typically *not as they should be, not as an inexperienced user would them expect to be*. For this reason we believe that the theorem proving community can learn from the computer algebra community. We will argue that the problem of the theorem proving systems is *not* just a deficiency of the interface, but that the problem with automated theorem provers is much deeper, it goes to the core of these systems, namely to the formal representation of mathematical knowledge and the reasoning that can be performed with this knowledge.

2 What is Good Design?

Donald Norman gives a fascinating introduction into "The Design of Everyday Things." His insights are of a very general nature and we will see that the principles for good design hold in mathematics as well.

Although Norman does not relate design to mathematics, most observations can be translated to a mathematical context. For instance, design principles allow us to answer questions like "Why and how do we find a proof without

[1] By the way, Gödel's proof that formal systems like the Principia are necessarily incomplete is irrelevant for this argument, since not only the Principia but any other powerful system suffers from the same problems.

major search effort, although it is a difficult one and we don't know it?" Norman states four principles why we get certain things right, although we don't know precisely what to do [Nor98, p.55]:

1. *Information in the world. Much of the information a person needs to do a task can reside in the world. Behavior is determined by combining the information in memory (in the head) with that in the world.*

2. *Great precision is not required. Precision, accuracy, and completeness of knowledge are seldom required. Perfect behavior will result if the knowledge describes the information or behavior sufficiently to distinguish the correct choice from all others.*

3. *Natural constraints are present. The world restricts the allowed behavior. The physical properties of objects constrain possible operations: the order in which parts can go together and the ways in which an object can be moved, picked up, or otherwise manipulated. Each object has physical features – projections, depressions, screwthreads, appendages – that limit its relationship to other objects, operations that can be performed to it, what can be attached to it, and so on.*

4. *Cultural constraints are present. In addition to natural, physical constraints, society has evolved numerous artificial conventions that govern acceptable social behavior. These cultural conventions have to be learned, but once learned they apply to a wide variety of circumstances.*

We argue that humans make use of such principles, not only in their relationship to everyday objects like door handles, but also in their relationship to mathematical objects like multiplication operators. Concretely, for these principles it means that mathematical objects are designed to bear information which makes them as easy to use as possible. Many proofs which we consider as trivial nowadays and which average students can find themselves now, were very hard when they were first discovered. Why do many things just fall into places and require only little search nowadays while at a time they were extremely difficult? We argue that this is due to a tremendous design effort that went into such a particular piece of mathematics in order to make the use of the related concepts as easy as possible. The aesthetic point that many mathematicians follow to present a proof in a form that is as easy and concise as possible, also pushes the practical limit of what can be proved further and further.

3 Design in Multiplication Tables

In this section we want to have a close look at one particular example of a mathematical concept which is carefully designed. Conventional theorem proving systems do support design issues only to a very limited degree. Multiplication tables form a concept that seems on a first view easy, and on a second difficult to model in existing theorem proving systems. Multiplication tables are part of rigorous mathematics in the sense that they appear not only as comments or

illustrations in mathematical textbooks, but are usually introduced in definitions and their properties are stated as theorems.[2] We believe that the features of the concept "multiplication table" as well as those we present in the next section are not only a matter of presentation but that they are used to encode and retrieve information about mathematical concepts in an efficient way and that they ease the actual process of finding and presenting proofs.

$\circ$	d_1	$\cdots$	d_n
d_1	c_{11}	$\cdots$	c_{1n}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
d_n	c_{n1}	$\cdots$	c_{nn}

Now let's look concretely at multiplication tables. They were first introduced by Cayley to represent the operation of finite abstract groups. The information encoded into the tables is that the operation is a binary operation, defined on $\{d_1, \ldots, d_n\} \times \{d_1, \ldots, d_n\}$ and with range $\{c_{11}, \ldots, c_{nn}\}$, the operation is discrete and has a finite domain and codomain.

The table has its own notion of well-formedness, that is, all d_i have to occur and have to be different, the table must be fully filled. Here you find Norman's principles 1, 3, and 4. Multiplication tables are designed in a way that their structure puts "information in the world" that makes it difficult to violate well-formedness. It is hard to imagine when you've got the task to define a specific operation starting with an empty multiplication table that you forget a case, since that would leave a hole in the structure. The table itself is of the form that it constrains the possibilities. For instance, it is impossible to enter more than one entry per field. This prevents any over-specification of $\circ$. Furthermore, although the order of the d_i in the columns and rows could in principle be different, cultural conventions prevent that.

Note that there are particular reasoning methods connected to the representation. To check the basic property of closedness, one has to go through the elements of the table and check whether for all elements holds $c_{ij} \in \{d_1, \ldots, d_n\}$. The commutativity of $\circ$ is checked by verifying that the table is symmetric with respect to the diagonal. This intuitive form of reasoning depends on the cultural convention to use the same order for rows and columns. Another cultural convention is to write a (potential) unit element as first element (or second element in the presence of a zero, which typically goes in the first place). With this convention, it is checked that d_1 is a unit element by establishing the equality of the columns under $\circ$ and d_1 and the rows right to $\circ$ and d_1. Inverse elements can be checked by establishing that each column and each row contain the neutral element exactly once. These reasoning patterns follow partly the design principle number 2, since they are natural and easy to reconstruct. From the group properties only associativity requires a logic level proof.[3]

Of course, it is possible to define the same operation in a logic, possible formalisations are for example:

[2] We use here the word "rigorous" and not "formal" in order to distinguish it from "formal logic" and mechanical systems. Mathematicians would probably use the word "formal," since they are happy with these concepts as a level of formalisation that clarifies concepts unambiguously.

[3] Surely, for people experienced with this type of proof, there isn't anything to prove anymore, but it boils all down to trivial computations, which – according to a principle which Barendregt calls the Poincaré principle [Poi02, BC01] – can be considered as not being a part of the proof. This is different for beginners, whose perspective we have taken here.

- First order: extend the signature by a function constant $\circ$ and add the assumptions $d_1 \circ d_1 = c_{11} \wedge d_1 \circ d_2 = c_{12} \wedge \ldots \wedge d_n \circ d_n = c_{nn}$.
- Higher order: use the description operator[4] to define the operation as $\circ \equiv \lambda x.\iota y.(x = (d_1, d_1) \wedge y = c_{11}) \vee \ldots \vee (x = (d_n, d_n) \wedge y = c_{nn})$.

While the special representation of multiplication tables can be translated into these general logical formalisms, parts of the information stored in the mathematical representation are lost. In the first case the compoundness of the table representation is hard to reconstruct. We are speaking about a set of equations suitable for equational reasoning steps, but to recognise that the set of equations is suitable for an abstract method for closedness or commutativity as mentioned before is not so obvious. Also the special reasoning methods for proving commutativity, inverse elements, and neutral element are not so obvious, but require search in a set of formulae. From a human interface point of view, the lack of structure in the formulae puts the burden of guaranteeing well-definedness on the human. He or she has to be careful not to forget the definition of one element or to over-define one expression by inserting two formulae like, for instance, $d_1 \circ d_1 = d_1$ and at a different place $d_1 \circ d_1 = d_2$.

Although, the higher order formalisation of the operation seems preferable over the first order one since it encodes $\circ$ as one compound object, here as well it is hard to recognise what kind of function is encoded. Actually, there is a proof obligation to be shown in an application of the function to arguments, namely that there is really a unique element with these properties.

If one decides to live with such limitations of the formalisation then complex concepts as multiplication tables can still be made available via an interface. Manipulations on the objects of the interface would have to be translated to the corresponding inference steps applied to the underlying formalisation. Besides the possible problems in the efficiency of such an implementation due to translation to the formalisation level, execution and retranslation, there seems to be a conceptional discrepancy. Why should the interface have more knowledge about the objects than is available in the representation language? When the interface has more knowledge about the object why should a manipulation on this level be translated to the lower level? The combination of different interfaces, e.g. when a complex concept contains other complex objects would have to be considered. The extension for new concepts would have to be implemented on two levels: for the interface and in form of formalisations (and for the interfaces of concepts related to the new object).

In our view, a better possibility is to add data structures for complex concepts to the object language of the logic. For multiplication tables this could be an array of size $n \times n$. In this case there is an object that corresponds to a multiplication table. However, depending on how this is done, another basic property may be missing: an array is not just a function. The functional behaviour has to be modelled by another definition for the application of arrays. It seems that logic is used as an interface, that is, logic is used to specify mathematical representations instead of offering them.

[4]The description operator ι returns the element of a singleton. $\iota y.P[y]$ denotes the unique element c such that $P[c]$ holds, if such a unique element exists.

This point to the necessity for a well-designed extension of the underlying formalism. There is no perfect formalism as can be seen by the arguments of the proponents for the different formal systems and the still ongoing search for 'the mathematical vernacular.' The extensions are usually general in the sense that they are applicable in many mathematical situations. Examples for extensions are subtypes, dependent types and/or a built-in recursion principle. They have all in common, that complex concepts are made primitive in the sense that proof obligations in the object logic are moved to specialised procedures, e.g. type-checking. Since it is not always possible to stay in the extended language, for instance, the primitives can become objects of mathematical theorems, there has to be a relativisation of primitives into the object language. A possible drawback is that the extensions do not force to make use of them. It is still possible to use formalisations that do not make use of the extension, which leads to different formalisations (probably incomparable in the object language) of the same concept in mathematics. We will discuss sorts as extension in section 5.1.

When we take together that different extensions are useful for different purposes, and that there exists a wide variety of complex concepts with specialised procedures in mathematics, then the introduction of new primitives and the possibility for their relativisation seem to be the basic principle for well-designed mathematical concepts, that is, implementations for mathematical concepts that have their basic features built-in.

While programming languages like Caml (see e.g., `http://caml.inria.fr/`) foster for the transition between different data structures, this is typically difficult in deduction systems. We will briefly discuss the little theory approach in Imps [FGT92] later (see section 5.2).

4 Mathematical Representations

In this section we look at further examples – in addition to the multiplication tables – for mathematical concepts and procedures which are difficult to represent directly in standard foundational systems. Although we do not relate them in detail to the design principles discussed in section 2, it wouldn't be hard to establish similar relationships here as well. We try to argue later that all these examples show that mathematical representations are very flexible and extendible, that new concepts may require new representations and that closed systems do not offer the necessary flexibility to design concepts to the level of sophistication which is achieved in informal mathematics.

4.1 Matrices

A matrix is a two dimensional array that may contain numbers or more complex objects. It has similarities with multiplication tables and a possible formal definition as lists of lists is the same as the one of multiplication tables. However its usage is very different, and human mathematicians have different methods attached to these concepts. Matrices are typically used to represent linear

mappings and other transformations in vector spaces. The equation[5]

$$\left(\begin{array}{c|ccc} \alpha & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & T^{-1} & \\ 0 & & & \end{array}\right) \cdot \left(\begin{array}{c|c} 1 & uT \\ \hline 0 & \\ \vdots & BT \\ 0 & \end{array}\right) = \left(\begin{array}{c|c} \alpha & \alpha uT \\ \hline 0 & \\ \vdots & T^{-1}BT \\ 0 & \end{array}\right)$$

is taken from a proof about the tridiagonalisation of matrices.

In principle matrices over a field F can be represented in logic by functions from an index set into F. However, this representation does not lend itself to the definition of multiplication of matrices in which product elements are calculated by traversing the left matrix left-right and the right matrix of the product top-down. The generalisation of this principle makes it easy to establish the relationship accounted for above. Furthermore the explicit matrix representation exhibits advantages mentioned above for multiplication tables.

In informal mathematics it is very easy to introduce a representation like matrices and then extend it to represent properties like the one in the equation above in a very concise form. Once properties about such a representation have been proved – like that the usual rule for the multiplication of matrices, also holds for matrices of matrices, provided the dimensions fit – objects in the new representation can be used as first class citizens to prove further properties – like the property expressed by the equation above. The extension of the multiplication rule to matrices of matrices makes actually the justification for the equation above very easy. For proving theorems, it is not necessary to go back to the low level descriptions that were originally used in order to define these matrices. Such a low level proof would actually be very hard to achieve from the high-level proof. In other words, if one wanted to view the high-level representation in the equation above as a mere interface matter, the interface would have to perform very complex procedures in order to link the simple high-level proof to a complicated low-level one.

Note that matrices are available in computer algebra systems as primitives and that direct manipulations are possible. This, however, does not make it obsolete to introduce them directly into automated theorem proving systems as well. For instance, the matrices used in the equation above cannot be easily defined in a computer algebra system, since they represent more than one instance, they are generalised matrices representing any matrix that has the same 'form.' Establishing the equation itself cannot be done by computation, but requires reasoning. Once established it can become a further computation rule.

4.2 Dynamic Representations

A feature of mathematical representations is that they are dynamic in the sense that as new knowledge is available the basic representation of objects may change. For instance, the existence of inverse elements of a *group* G can be

[5] Mathematicians store information even in the letters they choose for their objects. Without further information it is relatively easy to reconstruct that T^{-1}, B, and T should represent submatrices since they make use of upper-case letters, while the Greek α stands for a scalar, and u denotes a vector.

formalised by $\forall_{x \in G} \exists_{y \in G}\ x \circ y = e \wedge y \circ x = e$ with the unit element e. Since for each group element there exists exactly one inverse element, the inverse of an element x is usually denoted with the help of a function as $inv(x)$. Whereas the introduction of a function denoting the inverse elements is possible in most of the interactive theorem proving systems, the question whether the concept group should be introduced as $group(G, \circ)$, $group(G, \circ, e)$, $group(G, \circ, e, inv)$ seems to be more subtle.

The first formalisation eases the possibility to inherit properties of subsumed concepts with $group(G, \circ) \Rightarrow monoid(G, \circ)$. The latter formalisation makes the unit elements and inverse elements directly accessible but already contains the uniqueness of the inverse element of an element in form of the inverse function. The process of the actual exploration of basic principles of group theory that starts with the tuple $(G, \circ)$ and later introduces the neutral and inverse elements as useful parameters of the concept can hardly be modelled in current automated reasoning systems. Rather it is necessary to choose one formalisation and to stick to it. This way it is not only the case that the introduction of a concept cannot be modelled adequately, but perhaps more seriously re-representations of concepts are not supported. However, while one representation may be best suited for one task, it may turn out to be unsuitable for a different one. In the latter situation mathematicians change representations. Different formalisations model different views on something that is one single mathematical concept to a mathematician. If a mathematician had to use one of the standard theorem proving systems, he or she would need to know in advance which choice of representation to make, since the choice of a good formalisation is crucial for the success. But how do you know which formalisation is best, when you start to explore something?

Another example of dynamic re-representation, which is very simple, but for which the different reasoning complexities are striking, is when associativity holds for an operation $+$. Once associativity is established, no mathematician would still use brackets, but the notation for a term like $((1 + x) + y)$ would change to $1 + x + y$. The property is encoded into the notation and can be retrieved from the given term. On a reasoning level this simple shift in representation can make a dramatic difference. For a term with $n + 1$ summands there are $\frac{1}{n+1}\binom{2n}{n}$ different ways to put brackets in.[6] That means for a medium sized expression with just 10 summands there are already 4862 ways to represent it. If all these representations are part of the search process it is no surprise that automated theorem provers find such expressions difficult. The design of these systems does not allow for a change in representation, a user must write down unnecessary brackets in order to be syntactically correct, the brackets, however, don't help but unnecessarily confuse the reasoner. These all are signs of bad design. In the next sub-section we will look more closely at the change of representation.

[6] Even notation can be an object for mathematical investigations. The formula gives the Catalan numbers.

4.3 Change of Representations

Sometimes an appropriate reformulation of a problem into another representation is already the key step to find a proof. Different representations allow to apply knowledge from different sources to a problem. The importance of re-representation was pointed out by George Pólya [Pól62, vol.2, p.80]:

> *When you are handling material things (for instance, when you are about to saw a limb off a tree) you automatically put yourself in the most convenient position. You should act similarly when you are facing any kind of problem; you should try to put yourself in such a position that you can tackle the problem from the most accessible side. You turn the problem over and over in your mind; try to turn it so that it appears simpler. The aspect of the problem that you are facing at this moment may not be the most favourable: Is the problem as simply, as clearly, as suggestively expressed as possible? Could you restate the problem?*
>
> *Of course you want to restate the problem (transform it into an equivalent problem) so that it becomes more familiar, more attractive, more accessible, more promising.*

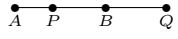
We exemplify this importance by different forms of re-representations:

- *functions:* Given an Euclidean space $\mathcal{R}$ with a metric function $|\ | : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ then for each pair $A, B \in \mathcal{R}$ of disjoint points there exists exactly one distance preserving function $g_A^B : \mathbb{R} \to \mathcal{R}$ with $g(0) = A$, $g(|AB|) = B$. With the help of this function the lines in Euclidean space can be interpreted as images of the real numbers. Notions of the real numbers, as intervals, correspond to notions in the abstract Euclidean space, namely line segments.

- *representation theorems:* Poincaré's model for the hyperbolic plane, where a line has infinitely many parallel lines through one point, is a unit disk, where circle segments correspond to hyperbolic lines. With this representation it becomes possible to re-represent constructions in the hyperbolic plane as constructions in the Euclidean plane.

- *theory change:* Geometric constructions can be represented as field extensions. The question of the possibility to construct a square equal in area to a circle using compass and ruler can be re-represented to the question whether π belongs to a particular class of algebraic numbers (which it does not since it is transcendental).

- *inheritance:* The properties of monoids and groups are inherited by the multiplicative and additive substructures of rings and fields.

- *diagrams:* $P \in [AB], B \in [AQ] \Rightarrow P \in [AQ], B \in [PQ]$ which is obvious when this situation is expressed in a diagram: 

And of course there exist re-representations between different theories. Some of them changed the structure of mathematics itself:

- *Cartesian geometry:* arithmetic representation for geometry. This makes it possible to reduce geometrical problems to arithmetic problems and solve them arithmetically. This is actually the starting point of Descartes' idea to take arithmetic as a foundational system so that all problems should be translated to arithmetic and then solved by equation solving.
- *set theory:* mathematical concepts are representable as sets. Set theory is another foundational system on which most of mathematics can be based *in principle*.
- *group theory:* the group concept can be used to represent geometric transformations, permutations, and the solvability of polynomials.

Certain forms of representations are very important to form new concepts. The theorem that every permutation can be decomposed into transpositions, that is, can be represented as a product of transpositions, makes the definition of even and odd permutations suggestive. It is hard to imagine how to define the concept without this particular representational form. Once these concepts have been formed they become the starting point for the introduction of other concepts like the alternating group.

Let's look at a different example, the concept of real numbers. Real numbers can be defined as Dedekind cuts or Cauchy sequences. However, Cantor's second diagonalisation proof that the reals are uncountable is difficult to imagine without having the representation in the decimal (dual, or another) number system.

Often the opposite of a change in the representation happens in mathematics, that is, so-called overloading is used. The use of the same notation for different concepts, e.g. $|\ |$ in the example above for the concept of distance, $\cdot$ for operations that behave like the well-known multiplication on natural numbers. $\cdot$ is used to represent multiplications between scalars and vectors, vectors and matrices, or matrices and matrices (as in section 4.1). Even formally incorrect notation, as equality for isomorphisms, is used to enable the transfer of knowledge from a known concept to a new concept. Certain *sort* and *type* systems allow for some kind of overloading, but they do not offer the kind of knowledge transfer that humans achieve this way in using overloaded symbols in an analogical way.

4.4 Structured Knowledge

A mathematical textbook is usually highly structured with *definitions*, *theorems* and *proofs* as main categories. This categorisation is reflected in the formalisations for deduction systems. A closer look reveals that there exists a finer classification. For instance, definitions can be simple, inductive, or implicit. Some theorems are explicitly introduced as tools which can be applied in other situations or contain equivalence statements that are useful for re-representation. A proof can contain subproofs that will be repeatedly used for

other theorems and that are emphasised to be important by the author, and a proof may contain an algorithm for the construction of mathematical objects.

A typical schema for the introduction of mathematical concepts is that a definition is followed by theorems giving simple properties and typical examples for this concept. One could argue that this structure has no significance for deduction systems because it is solely beneficial for the human process of learning and understanding. We try to show that the structure can become important as a basis for the reasoning process.

Take as an example the continuity of functions. The basic properties that are usually provided after this definition are that continuity is preserved for the sum, product, and composition of continuous functions. Usually as an example the identity on the reals will be given. Now suppose it is required to show that any polynomial is continuous. Instead of going back to the definition itself, the basic properties of the concept and the example of the identity function are used to prove the continuity of polynomials. The basic properties can be seen as attached to the definition and preferably used to prove simple proof obligations.

Of course it is possible to argue that the attached properties can be retrieved from a database that consists only of the plain structure of definitions and theorems. However, it is then difficult to query for a property like continuity of $+$. Should the query consist of all theorems containing the symbol 'continuous', the symbol '$+$', both, or any expression that is equivalent to the definition of the concept? The structure that is lost in the plain logic representation would have to be reconstructed through elaborated query techniques that give useful results.

Note that the standard mathematical practice to build hierarchies of mathematical concepts is a very important means to reduce complexity in theorem proving. For instance in set theory, it is possible to define symbols like $\subset$, $\cup$, $\cap$ by $\in$. When you build a topology on top, using functions like $\circ$ for inner and $^-$ for closure, you can carry through most proofs on a level where you make use of abstract properties of $\subset$, $\cup$ and $\cap$ and can avoid totally to go down to the level of elements that involves $\in$.

Another category of mathematical knowledge form examples. They are crucial for the understanding of concepts. This is not only important to give semantics to syntactically defined concepts, but it can also be very relevant on the level of syntactic proof construction. We exemplify this for the theorem that "Groups G of order greater than two have non-trivial automorphisms." It seems to be hard to synthesise automorphisms directly only from the given assumption that G is a group. In this case, the standard human heuristic to try one of the typical examples, $x \mapsto gxg^{-1}$ and $x \mapsto x^{-1}$, provides already the crucial idea to prove this theorem.

4.5 Limited Coverage of Mathematical Activities by Logic

When we take mathematical textbooks as basis for what is part of mathematics and what not, we can find statements that are not expressible in formal languages at all. Naturally comments or diagrams, and a number of models are

not represented. Historic statements, statements about the relevance of properties and concepts and so on are part of mathematics, but not part of logic, although they are often important for a deeper understanding and an informed proof search.

But even at the level of problem formulation, it is not always possible to apply a formal system in a straightforward way. Let's look for instance at the mathematical task (provided in some context and for a concrete function f): "Determine the maximum of $f(x)$ in the interval $[a, b]$." Since the task ask for the value it is not clear whether the obvious formalisation "$\exists_{x\in[a,b]}\ \max(f, x)$" reflect the meaning of the sentence. Assume somebody comes up with the argument "Since f is a continuous function on a compact interval it has a maximum." This might be a correct argument to prove the logical formulation but it would not solve the original task. On a closer look adequacy of the logical statements depends on the logic used. If interpreted in a constructive way the logical formulation is adequate; if interpreted classically it is not. While there are constructive and classical systems around, it seems to be inappropriate that one has to decide for a constructive system once and for all, in order to be able to formulate a standard task like the one above. In mathematical practice one wants to know a concrete value x_0, but the proof that the function has a maximum at x_0 is typically done classically and not constructively.

While one can hardly cover all aspects of mathematics in a computer-based system, it seems for many applications – like the recently emerging applications in education – important to find a coverage which is as broad as possible.

4.6 A 'Natural' Calculus

The suggestions for a 'Mathematical Vernacular' [Bru94] seem not to question that all concepts of mathematics are *sufficiently* expressible in a language consisting of functions and relations, potentially enriched by types or sorts. The design of the Vernacular seems not to be focused on the objects mathematicians are interested in, but on the reasoning framework.

The strength of a formal system can be measured by the de Bruijn factor, that is, the ratio of the length of the proof in the formal system compared to its version in a mathematical textbook.[7] A reassuring observation is that in the experiments with Automath and Mizar the de Bruijn factor remained constant for the proofs of a wide range of differently complex theorems.

Even if we agree with the conclusion that the proofs constructed in formal calculi are already in principle an approximation of standard mathematical proofs, it is important to note that such a comparison is based on the *output* of mathematical work, the language used by mathematicians to communicate proofs in textbooks and articles. While a comparison of such completed proofs, proofs after all search is finished may be interesting, they often do not resemble the proof construction. As an example look at standard ϵ-δ-proofs. In the proof construction you compute a sufficient criterion on δ, choose δ as a function of ϵ

[7] The notion is not without problems, since mathematical proofs are not standardised with respect to their detailedness. Furthermore there are other aspects for the quality of a proof than its length.

and than you prove that with this δ the difference of the function values is indeed smaller than ϵ. In a finished proof a crucial part of the proof construction, namely the construction of δ, is redundant and hence not presented. For this reason it is impossible to understand prima facie why δ has been selected as it is.

Traditionally, the formulations of final proofs are minimalist and mathematicians repress their original ideas, even the order of steps can be different, in favour of an objective rigorous style. As Pólya [Pól54, p. vi], pointed out:

> *We secure our mathematical knowledge by demonstrative reasoning, but we support our conjectures by plausible reasoning ... Demonstrative reasoning is safe, beyond controversy, and final. Plausible reasoning is hazardous, controversial, and provisional. ... In strict reasoning the principal thing is to distinguish a proof from a guess, a valid demonstration from an invalid attempt. In plausible reasoning the principal thing is to distinguish a guess from a guess, a more reasonable guess form a less reasonable guess. ... [plausible reasoning] is the kind of reasoning on which his [a mathematician's] creative work will depend.*

The 'demonstrative reasoning' corresponds to a formulation in reasoning steps that were investigated by logicians. In this sense current deduction systems are suitable as proof checkers for existing and well-understood parts of mathematics but lack to act as proof assistants for the exploration and construction of new mathematical knowledge. How can 'plausible reasoning' be modelled? Proof planning follows the paradigm of proof search on an abstract level and can be seen as an important step into this direction. But as described in [BMM$^+$01] even proof planning depends on structural restrictions of the underlying calculus and uses a formal language as the only representation for mathematical concepts.

There might be the view that we can come up with a more powerful logic which provides the best possible representation. Marvin Minsky gave a strong argument why this wouldn't be the case in artificial intelligence in general, but why multiple representations are necessary. He recommends [Min92]:

> *Everywhere I go I find people arguing about which representation to use. One person says, "It is best to use Logic." The next person says, "No, logic is too inflexible. Use Neural Networks." The third person says, "No, Neural Nets are even less flexible, because you have to reduce everything to mere numbers. Instead, you should use Semantic Networks. Then, the different kinds of things can be linked by concepts instead of mere numbers!" But then the first person might complain, "No, Semantic Nets are too arbitrary and undefined. If you use Formal Logic, that will remove those ambiguities."*
>
> *What is the answer? My opinion is that we can make versatile AI machines only by using several different kinds of representations in the same system! This is because no single method works well for*

> *all problems; each is good for certain tasks but not for others. Also different kinds of problems need different kinds of reasoning. For example, much of the reasoning used in computer programming can be logic-based. However, most real-world problems need methods that are better at matching patterns and constructing analogies, making decisions based on previous experience with examples, or using types of explanations that have worked well on similar problems in the past. How can we encourage people to make systems that use multiple methods for representing and reasoning? First we'll have to change some present-day ideas. For example, many students like to ask, "Is it better to represent knowledge with Neural Nets, Logical Deduction, Semantic Networks, Frames, Scripts, Rule-Based Systems or Natural Language?" My teaching method is to try to get them to ask a different kind of question."First decide what kinds of reasoning might be best for each different kind of problem – and then find out which combination of representations might work well in each case."*

As we have seen for multiplication tables different types of representations allow for specialised and efficient reasoning methods connected to them, opposed to search on the logic level. Mathematicians carefully design their concepts to keep the search spaces small. We need to understand this aspect of mathematical reasoning much better in order to understand the versatility of the reasoning capabilities of human mathematicians.

Historically the development of many concepts went the way that certain meta expressions were introduced, which were later reified and become object expressions. The development of number systems might illustrate this. Having natural numbers and fractions, irrational numbers and negative numbers as well were considered as odd ones out, which only later became first class citizens. Then imaginary numbers were the odd ones and negative square roots were considered as strange entities which were used only for convenience. Likewise, functions were first concrete in nature, and only much later it was possible to speak about abstract function spaces.

5 Representations of Mathematics on Computers

Up to here we have strongly argued how important a broad range of specialised constructs is for the adequate representation of mathematical knowledge and for proof construction. Representations find their analogues in data structures used in existing implementations of mathematical software systems, a range of general purpose and specialised theorem proving systems, computer algebra systems, and educational software. These data structures provide functionality and the ability to implement mechanisms working on them. In this section we want to take a brief look at some important features of systems from which ideas can be borrowed to realise a flexible system of the kind we envisage.

5.1 Sorted Extensions to Logic

Sorted logic is a good example to exemplify the importance of careful design. It is an old insight that sorted logic (more carefully put, some classes of sorted logics) can have significant advantages over unsorted logic. Let us just consider the case of standard first-order logic with and without its order sorted extension. Sorts can be considered as special unary predicate symbols. Typical formulations in sorted logic are $\forall x_{\mathsf{Human}}.\mathsf{Mortal}(x)$, $\mathsf{socrates} \leqslant \mathsf{Human}$, $\neg\mathsf{Mortal}(\mathsf{socrates})$. The equivalent in unsorted logic is $\forall x.\mathsf{Human}(x) \Rightarrow \mathsf{Mortal}(x)$, $\mathsf{Human}(\mathsf{socrates})$, $\neg\mathsf{Mortal}(\mathsf{socrates})$. The translation from the sorted to the unsorted logic is called relativisation. The possibility to relativise any sorted problem formulation can be and is used as an argument *against* the use of sorted logic in line with the standard argument "Everything you can do in your system of *sorted logic*, I can do in *unsorted logic*." This argument is – although correct – unhelpful because of the counterargument "Everything you can do in your *unsorted logic*, I can do in *sorted logic*, but in a smaller search space!"

The reason for the smaller search space is due to a better design realised by the sorted system. Although the formalism of sorted logic is equivalent in strength to the unsorted one, a concrete formulation (which makes use of sorts in a non-trivial way, that is, whose relativisation is not equal to itself) is *not*. It is actually *weaker*, since certain things can *not* be derived in sorted logic which can be derived in unsorted logic (concretely, formulae like $\mathsf{Human}(1) \Rightarrow \mathsf{Mortal}(1)$ are syntactically possible and can be derived in unsorted logic, but the equivalent is rejected in sorted logic). To generalise this observation: A particular design is *better* than an alternative one if certain redundant, or heuristically uninteresting derivations cannot be made.

The integration of sorts in a system also demonstrates how subtle design issues can be. We can't discuss this in detail here, but we will give some indication. It should be noted that sorts are not necessarily exclusively beneficial.[8] If we take standard order-sorted logic we have for each relation like Human and Mortal to decide whether we want to formalise it by a sort symbol or by a predicate symbol. If we want to prove some statement like $\neg\mathsf{Human}(\mathsf{pegasus})$, we cannot formalise Human as a sort symbol. This is a serious flaw in standard sort systems, also it is not possible to model the genesis of concepts in an adequate way. In order to perform the reasoning above we need to know $\mathsf{socrates} \leqslant \mathsf{Human}$ a priori. It is not possible to infer that Socrates is a human being later in the reasoning process. This unduly limits the flexibility of the system. Ideally you would want to have the advantages of a sorted formulation whenever possible, but also benefit from the flexibility of the unsorted formulation when necessary. To our knowledge only Christoph Weidenbach's system [Wei93] offers these possibilities.

5.2 Little Theories

Another important logic-based approach which approximates mathematical reasoning well is the little theory approach of Imps [FGT92]. A theory given by

[8] For a detailed discussion why sorts/types may be harmful see [Lam95].

a signature and a set of axioms can be imported into another theory, if there exists a theory interpretation which maps the constants and axioms of a source theory to expressions and theorems of a target theory. With this approach theorems can be reused in other theories and new theories can be defined as combination of existing theories.

As we can see in textbooks from different areas of mathematics, there is no fixed hierarchy of theories. Each textbook presumes knowledge from different areas of mathematics and by this induces a partial order of theories. That there exists a total order of theories is rather a theoretical result than used in everyday mathematics. The attempt to realise this total order prohibits the flexibility to use theorems constructed for one area in another area. The little theory approach also allows to relate different formulations with each other without necessitating a hierarchy.

5.3 Data Structures in Computer Algebra Systems

Computer algebra systems offer many more primitive data structures such as matrices than standard theorem proving systems. With these structures it is possible to cover large parts of the 'computational' part of mathematics. As we tried to show, there is a grey area in which deduction and computation go hand in hand (cf. sub-section 4.1), since any structure in a computer algebra system is concrete, while mathematical expressions often make use of ellipses, for instance, expressions which contain dots like $x_1, x_2, \ldots, x_n$, or the multiplication table and matrix in sections 3 and 4.1. A promising first approach in the direction of formalising ellipses in reasoning can be found in [BR99].

5.4 Data Structures for Reasoning

Koedinger and Anderson [KA90] introduce a representation different from a purely logical formalisation called diagram configuration model (DC) for diagrammatic reasoning in geometry. The representation is based on DC schemas that encode typical geometric situations that were identified through observations on the problem solving behaviour of experts in this domain. The schemas contain the main property of the situation, the subsumed properties and the different ways under which the schema can be established. The level of abstraction allows for an efficient inference algorithm that introduces the DCs as inference steps.

Mateja Jamnik describes in [Jam01] a diagrammatic representation for theorems and inference steps based on this data structure that allows to infer theorems in arithmetic. The proofs constructed in the diagrammatic representation can be translated to proof planning and then formally verified with a theorem prover.

These examples are important in our context since they show that at least for particular domains it is possible to build special reasoners for diagrammatic reasoning which are distinct from a purely logic based system, but which can be formally linked to such a system. An adequate data structure for diagrams seems to be the key point for the success of both approaches. Only this data structure allows the implementation of an efficient inference mechanism.

6 Discussion

One of the deepest insights in the foundations of computing was Turing's paper on computable numbers, which clarified by a construct which we now call a Turing machine what can and cannot be computed by computers. This does not mean, however, that the core field of Computer Science would offer to users just Turing machines in which they have to write their programs and if they don't like the idea tell them that everything they may possibly want to do with computers can be written as a Turing machine. It is not even the case that computer languages are built as extensions of Turing machines or compiled into Turing machines. The field of automated theorem proving seems to have followed a different approach. While it is built upon the deep logical insights of the first half of the 20th century, the rich wealth of structure and representations in mathematics has not been mirrored sufficiently yet in formal systems.

In this paper, we presented aspects of mathematical representations that we think are highly relevant for mathematical theorem proving and that can – if at all – be implemented in traditional theorem proving systems only with great difficulty. We think that the lack of acceptance of theorem proving systems among mathematicians (compared with the success of computer algebra systems) is also due to these shortcomings. Deduction systems offer – compared to computer algebra systems – often only limited added value: formal correctness, but at a very high price, namely a significant overhead to formulate and prove mathematics. While the production of a large body of formally correct proofs for known theorems does usually not correspond to the research interests of mathematicians, the exploration of new problems is not very well supported as we observed in section 4.

Current deduction systems ideally offer the potential benefit that they can relieve users from the tedious task of checking trivial subproblems, so that the human can concentrate on the interesting parts of the problem. In practice, however, it is typically more work even for an experienced human mathematician to formulate the problems in the first place so that an automated theorem prover can prove them than to do the job directly him/herself. We think, when a theorem proving system is to have a real application as either a proof assistant or a proof tutor, it has to take care of the representation that is used by mathematicians.

The mathematical representation is not only important for the user interface and presentation of proofs, but for theorem proving itself, since the representation is used to optimise problem solving and the transfer and access of knowledge. The observations presented in section 4 reveal that mathematical knowledge is highly structured and special representations are important for mathematical problem solving. Data structures that realise these aspects may allow for the definition of detailed problem classifications for which special but efficient mechanisms can be found. In sub-sections 5.1 and 5.4 we gave examples for data structures and implementations of this kind. Mathematical representation appears furthermore to be dynamic and flexible. Modelling this flexibility is on the one hand important for the user interface, because it would give the choice of representation to the user. On the other hand, this flexibility

is the key point for the combination of different representations.

We have summarised some approaches to good design in section 5. More work in this direction is necessary to offer well-designed tools to mathematicians and other people interested in computer assisted mathematics. While certain parts might turn out to be straightforward other aspects will require a much deeper understanding. To give one example for the size of the task, if we wanted to integrate multiplication tables as a primitive in automated reasoning systems, it should be relatively easy to offer concrete multiplication tables (or matrices) with all the advantages mentioned in sections 3 and 4 (as it is relatively easy to offer order-sorted sorts). However, it will be difficult to offer general type multiplication tables (or matrices) which contain ellipses, and flexible ways to reason about them. Human beings have an amazing capability to reify structures in their space of discourse. We seem not to have yet a deep understanding of these capabilities.

This paper compared approaches in mathematics and in existing theorem proving system. We were with it not so ambitious to try to offer solutions for the deficiencies of existing systems, but we want to conclude with some thoughts how a strong reasoning system could look like to model mathematical practice more adequately. We think that our examples indicate that it will be impossible to build a system with a fixed syntax which incorporates all possible types of mathematical structures. Rather we think that a system should be extendible by the user. This puts a very important principle at risk, namely the simplicity of a proof checker that can check proofs generated with the system (called de Bruijn principle in [BC01]). It is important since it means that the correctness of proofs does not depend on the (currently not achievable) correctness of a very big and complex system for proof construction, as long as there is a small reliable system that can check generated explicit proof objects. One wouldn't want to sacrifice this principle light-heartedly, since much of the motivation (at least in some of the expectations as laid out in the introduction) for using computer-based systems would be at stake. We think, however, that it should be possible to develop a system which is flexible and extendible in the sense we discussed *and* allows for an only slightly more complicated checker. We imagine that this can be achieved by providing explicit semantics to each extension, which founds the extension in the original system.

While the investigation of such a system remains future work, we imagine that it should firstly be its own meta-system, so that it is possible to use its representations as starting point for the formation of new concepts (care about paradoxes has to be taken). Secondly, a flexible system for the definition of mathematical structures as well as establishing their formal relationships would have to be provided by such a system. Thirdly, a graphical user interface should facilitate the possibility to relate mathematical structures to a form, which is familiar to mathematicians. Particularly important are in this context spatial and diagrammatic representations. While there are approaches to all these points, to our knowledge no single system offers yet an integrated flexible approach.

Acknowledgements

We thank Mateja Jamnik for fruitful discussions and suggestions. Martin Pollet thanks Peter Pröhle for the insights he gave him into the process of mathematical problem solving. Furthermore we thank the Calculemus community for valuable feedback.

References

[BC01] H. Barendregt and A.M. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, **32**(5):3–22, 2001.

[Bee98] M. Beeson. Automatic generation of epsilon-delta proofs of continuity. In J. Calment and J. Plaza, editors, *Artificial intelligence and symbolic computation*, pages 67–83. Springer Verlag, Berlin, Germany, LNCS 1476, 1998.

[BL85] R. Brachman and H. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Los Altos, California, 1985.

[BMM+01] C. Benzmüller, A. Meier, E. Melis, M. Pollet, J. Siekmann, and V. Sorge. Proof planning: A fresh start? In M. Kerber, editor, *IJCAR-Workshop: Future Directions in Automated Reasoning*, pages 30–37, Siena, Italy, 2001.

[BR99] A. Bundy and J. Richardson. Proofs about lists using ellipsis. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th LPAR*, pages 1–12. Springer Verlag, Berlin, Germany, LNAI 1705, 1999.

[Bru94] N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. Nederpelt, J. Geuvers, and R. de Vrijer, editors, *Selected Papers on Automath*, pages 865–935. Elsevier, North-Holland, Amsterdam, The Netherlands, 1994. Studies in Logic, Volume 133.

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. of the 9th CADE*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany, LNCS 310.

[Bun91] A. Bundy. A science of reasoning. In *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Elb98] H. Elbers. *Connecting Informal and Formal Mathematics*. PhD thesis, Eindhoven University of Technology, 1998.

[FGT92] W. Farmer, J. Guttman, and F. Thayer. Little theories. In D. Kapur, editor, *Proc. of the 11th CADE*, pages 567–581, Saratoga Springs, New York, USA, June 1992. Springer Verlag, Berlin, Germany, LNAI 607.

[Gol84] R. Goldblatt. *Topoi: The Categorial Analysis of Logic.* Elsevier, North-Holland, Amsterdam, The Netherlands, 1984. Studies in Logic, Volume 98.

[Hay74] P. Hayes. Some problems and non-problems in representation theory. In *Proc. of the AISB Summer Conference*, pages 63–79, Sussex, 1974.

[Jam01] M. Jamnik. *Mathematical Reasoning with Diagrams: From Intuition to Automation.* CSLI Press, 2001.

[KA90] K. Koedinger and J. Anderson. Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14:511–550, 1990.

[Lam95] L. Lamport. Types are not harmless. Technical report, 1995.

[Mar99] U. Martin. Computers, reasoning and mathematical practice. In U. Berger and H. Schwichtenberg, editors, *Proceedings of the NATO Advanced Study Institute on Computational Logic, Marktoberdorf, Germany, July 29 - August 10, 1997.* Springer Verlag, 1999.

[Min92] M. Minsky. Future of AI technology. *Toshiba Review*, 1992. `http://web.media.mit.edu/~minsky/papers/CausalDiversity.txt`.

[Nor98] D. Norman. *The Design of Everyday Things.* The MIT Press, London, 1998.

[Pel91] F.J. Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proc. of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufmann, San Mateo, California, USA.

[Poi02] H. Poincaré. *La Science et l'Hypothèse.* Flamarion, Paris, France, 1902.

[Pól54] G. Pólya. *Mathematics and Plausible Reasoning.* Princeton University Press, New Jersey, USA, 1954.

[Pól62] G. Pólya. *Mathematical Discovery – On Understanding, Learning, and Teaching Problem Solving.* Princeton University Press, New Jersey, USA, 1962.

[Rus03] B. Russell. *The Principles of Mathematics.* George Allen & Unwin Ltd, London, UK, 2nd edition, 1937 edition, 1903.

[Rus18] B. Russell. The philosophy of logical atomism. *The Monist*, 28/29:495–527/32–63,190–222,345–380, 1918. republished in [Rus56, p.177-281].

[Rus56] B. Russell. *Logic and Knowledge.* Allen & Unwin, London, 1956.

[Wei93] C. Weidenbach. Extending the resolution method with sorts. In *Proc. of the 13th IJCAI*, pages 60–65, Chambéry, France, 1993.

[Wit56] L. Wittgenstein. *Remarks on the Foundations of Mathematics.* Basil Blackwell, Oxford, England, third edition, 1956.

[WR10] A.N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge University Press, Cambridge, UK, 1910.