Warsaw University - Białystok Campus
Department of Logic
Liniarskiego street 4
15-420 Białystok, Poland

The QED Workshop II

held in Warsaw

July 20 – 22, 1995

edited by Roman Matuszewski [1]

Technical Report No. L/1/95

October 1995

The workshop was under the auspices of the State Committee for Scientific Research (Poland) - grant OGT 42/95, supported by special funding from the Office of Naval Research (USA) under ONR Order No. N00014-95-M-0072, cosponsored by Microsoft (Poland) and the Mizar Users Group.

[1] To order this report please write to `romat@plearn.edu.pl` .
The report is also available on the Web `http://www.mcs.anl.gov/qed/index.html` .

Contents

Introduction

Abstract

On July 20 - 22, 1995, Warsaw University (Bialystok Branch) hosted the QED Workshop II. The workshop was under the auspices of the State Committee for Scientific Research (Poland), supported by special funding from the Office of Naval Research (USA), cosponsored by Microsoft (Poland) and the Mizar Users Group.

QED is the title of an international project to build a computer system that effectively represents much of important mathematical knowledge and technique. The QED system will conform to the highest standards of mathematical rigor, including the use of strict formality in the internal representation of knowledge and the use of mechanical methods to check proofs of the correctness of all entries in the system. A principal application of the QED system will be the verification of computer programs. For background on the idea of the QED Project see "The QED Manifesto", Automated Deduction - CADE 12, Springer Verlag, LNAI 814, pp. 238-251, 1994, and available by anonymous ftp at info.mcs.anl.gov, file pub/qed/manifesto. The results of the QED Workshops are documented in URL http://www.mcs.anl.gov/qed/index.html .

The workshop was split into one-hour discussions dedicated to specific problems. Each such discussion was preceded by a short introductory lecture.

The QED Workshop II, held in Warsaw on 20–22 July 1995, was the second meeting devoted to the inchoate QED project. Robert Boyer and Andrzej Trybulec were responsible for the scientific programme, with Roman Matuszewski being the active chair.

The overall idea of QED is best described by the preamble to the *QED Manifesto* - an anonymously authored document that discusses the desirability and feasibility of organizing a proof-checked encyclopedia of mathematics:

> "QED is the very tentative title of a project to build a computer system that effectively represents all important mathematical knowledge and techniques. The QED system will conform to the highest standards of mathematical rigor, including the use of strict formality in the internal representation of knowledge and the use of mechanical methods to check proofs of the correctness of all entries in the system.
>
> The QED project will be a major scientific undertaking requiring the cooperation and effort of hundreds of deep mathematical minds, considerable ingenuity by many computer scientists, and broad support and leadership from research agencies."

The first QED Workshop was held at Argonne National Laboratory, on May 18–20, 1994. The most important conclusion of that workshop was that QED was an idea worthy pursuing. The majority of participants of the Warsaw workshop subscribed to that conclusion entirely.

The purpose of the workshop was to assemble a group of researchers to further pursue the idea of building a mechanically proof-checked encyclopedia of mathematics. The workshop was run as a sequence of one hour long discussions each devoted to a specific QED-related topic. The topics, which were suggested earlier by the participants and accepted for presentation, varied from very general issues of sociological and political nature to specific technical questions. A short summary of each of the presentations and discussions is attached.

Therefore, the workshop took on the form of a discussion meeting rather than that of a conventional conference. In the numerous discussions a distinction was made between:

(i) the software that organizes and administers the database of mathematical knowledge, and

(ii) the system or systems that are used to demonstrate the correctness of the results in the first place.

A clear short term goal was identified: we should attempt making the results of various systems publicly available, preferably on the World Wide Web. Then, over the course of time, we can investigate more sophisticated ways of integrating the diverse range of the already accumulated, machine readable mathematical knowledge.

The workshop was attended by 28 researchers, from 9 countries, representing ongoing world-wide efforts in theorem proving and mathematics. 11 of the participants attended the QED Workshop I at Argonne in 1994. The list of participants is given below:

Australia
Rajeev Gore (rpg@cisr.anu.edu.au)

Canada
Małgorzata Korolkiewicz (mkorolki@vega.math.ualberta.ca)
Bill Pase (bill@ora.on.ca)
Piotr Rudnicki (piotr@cs.ualberta.ca)

Estonia
Rein Prank (prank@cs.ut.ee)

Germany
Bernd Ingo Dahn (dahn@mathematik.hu-berlin.de)
Wolfgang Jaksch (Wolfgang.Jaksch@informatik.uni-erlangen.de)
Manfred Kerber (kerber@cs.uni-sb.de)
Herbert Stoyan (Herbert.Stoyan@informatik.uni-erlangen.de)
Martin Strecker (strecker@informatik.uni-ulm.de)
Claus Zinn (Claus.Zinn@informatik.uni-erlangen.de)

Japan
Yozo Toda (yozo@aohakobe.ipc.chiba-u.ac.jp)

Poland
Grzegorz Bancerek (bancerek@impan.gov.pl)
Roman Matuszewski (romat@plearn.edu.pl)
Bogdan Nowak (bnowak@krysia.uni.lodz.pl)
Stanisław Spież (spiez@impan.gov.pl)
Andrzej Tarlecki (tarlecki@mimuw.edu.pl)
Andrzej Trybulec (trybulec@cksr.ac.bialystok.pl)

Russia
Oleg Okhotnikov (okhezinv@math.urgu.e-burg.su)

UK
John Harrison (John.Harrison@cl.cam.ac.uk)

USA

Robert Boyer (boyer@cli.com)
John McCarthy (jmc@sail.stanford.edu)
William McCune (mccune@mcs.anl.gov)
Randall Holmes (holmes@math.idbsu.edu)
Paul Jackson (jackson@cs.cornell.edu)
Deepak Kapur (kapur@cs.albany.edu)
Javier Thayer (jt@linus.mitre.org)
Peter White (peter@opus.geg.mot.com)

The participants of the workshop shared the opinion that the final shape of QED will be achieved through a long sequence of small evolutionary steps. The starting point of this evolution is formed by existing theorem provers and proof-checkers, especially the ones that have already accumulated sizable data-bases of machine checked mathematics.

We would like to mention an initiative of John McCarthy who presented a talk on "Heavy Duty Set Theory" in which he gave examples of inferences which he felt should be regarded as (mathematically) "obvious" to a practical proof checker. He challenged the participants to replicate his solution of the mutilated checkerboard[1] in various systems and then compare how far the solutions are from his expectations. As far as I know two systems met the challenge. One of these is published in this Report[2]. For another mutilated checkerboard mechanical checking please see `ftp://ftp.cli.com/pub/nqthm/nqthm-1992/examples/subramanian/mutilated-checkerboard.ps`.

There were two general discussions. The first, a panel discussion led by Piotr Rudnicki, considered general aspects, as seen by the 8 panelists (G. Bancerek, M. Korolkiewicz, B. McCune, B. Pase, R. Prank, S. Spież, H. Stoyan, and A. Tarlecki). The panelists expressed both their enthusiasm about specialized QED subsystems and reservations about being too optimistic too soon. In particular, there was some discussion about how to attract mathematicians to the initial steps of building QED. There were no doubts that when QED is sufficiently developed, mathematicians will be glad to use it.

The second general discussion, led by Deepak Kapur, concerned general views on the future of the QED Project (the next workshop, the need of common meta logic, short term goals, what should be the administrative structure managing the QED data base, and how to achieve a steady flow of contributions to the QED mailing list). The participants expressed their hope that the next QED Workshop will happen in 1996.

Roman Matuszewski[3]
Workshop Chairman

[1] *The Mutilated Checkerboard in Set Theory* (see pp. 25 – 26).
[2] *The Mutilated Chessboard Problem* (see pp. 43 – 45).
[3] Warsaw University, Białystok Campus, Department of Logic, Liniarskiego street 4, 15-420 Białystok, Poland, `romat@plearn.edu.pl`

Can we Resolve the Tension between Constructive Type Theorists and Classical Mathematicians?

Paul B. Jackson[1]

1 Introduction

Constructive type theories (CTTs) are advocated as a foundation for mathematics which replaces classical logic and set theory. Significant work has gone into building interactive theorem-proving systems based on CTTs [dB80, C+86, Jac95, AGNvS94, LP92, CCF+95] and it seems desirable to involve the projects currently around such systems in any future QED venture. However, mathematics based on CTTs is rather different from the usual classical mathematics taught in schools and universities. QED must support this classical mathematics if it is to have any success. How then is cooperation possible?

2 CTT-based mathematics

All CTT-based mathematics has a computational reading. For example, a theorem of form:

$$\forall x \in A. \exists y \in B. P_{x,y}$$

can be read as saying that given any x in set A, there is a method of *computing* a y in set B satisfying the predicate $P_{x,y}$. Further, a CTT proof of such a theorem precisely specifies one such method. In general, theorems in CTT-based mathematics can be considered as specifications for programs and proofs as guides for the automatic construction of programs. This program synthesis paradigm has been one of the major factors stimulating recent interest in CTTs.

CTT-based mathematics so far has been largely modelled on the school of constructive mathematics first developed by Bishop [BB85, MRR88]. A feature of the Bishop school that increases its acceptability to classical mathematicians is that every theorem also has a reading as a classical theorem. This is not the case with other schools of constructive mathematics.

CTT-based mathematics has a finer grain than classical mathematics. Many distinctions are made that offer alternative computational readings. These distinctions are at a very basic level: for example, different readings are frequently be given for equivalence and subtype relations. It is a challenge to decide which alternatives to adopt and to keep the number of alternatives considered to a reasonable size.

Formalization forces definite choices to be made on alternatives where in the texts the need for a decision is glossed over or delayed. It also frequently turns out that functions need extra arguments that provide computational information. A formal development must include these arguments, though they also are often glossed over in the texts.

Current CTTs are somewhat awkward. In nearly all, the notion of *type* is not nearly as versatile as that of *set* in set theories. For example, equality of types is usually not extensional

[1]e-mail: `pbj@dcs.ed.ac.uk`

and a principle of comprehension is usually lacking. Also, many CTTs are regarded as being too complex to be acceptable as foundational theories.

Examples of formalization of mathematics in CTTs include the intermediate value theorem and some basic abstract algebra [For93, Jac95].

For the above reasons, formalizing Bishop-style mathematics in CTTs seems to be a significantly slower and more uncertain process than formalizing classical mathematics.

3 Opportunities for Cooperation

3.1 Libraries

Sharing of libraries on elementary concrete topics such as integers and finite sequences (lists) might be possible since there CTT-based and classical mathematics are similar. However, sharing of libraries on more abstract topics would be much more problematic.

Importing of classical developments into a CTT setting would be all but impossible because all the essential distinctions would be missing.

Importing a CTT-based development into a classical setting is possible, though the classical mathematician is likely to consider all the extra distinctions as irrelevant clutter. More pragmatically, the need to import might not be there, since the quantity and sophistication of formalized classical mathematics is likely to be much greater than that of CTT-based mathematics, both for reasons given in the previous section and simply because the corpus of formalizable classical mathematics is far far larger.

3.2 Systems Development

Opportunities are brightest here. There are many engineering challenges common to any system intended for helping to develop mathematics. For example, in the areas of user interfaces, mathematical databases, and automated reasoning.

3.3 Classical Mathematicians using CTT-based Systems

It has been shown consistent to extend CTTs with oracles in order to create classical type theories [How91]. A CTT-based system with such an extended CTT could be used by a classical mathematician to develop classical mathematics, though work is still needed to demonstrate that the extended type theory would have a versatility approaching that of set theory.

3.4 Constructive Type Theorists using Classical Systems

It might be possible to persuade the architects of CTT-based systems to consider seeking in a classical system some of the advantages they attribute to CTTs.

For example, type theories are claimed to provide a more structured language than set theory for mathematics, closer to that used in normal mathematical practice. However the Mizar project [Rud92] has demonstrated how such advantages can be gained by layering a type system on top of a classical set theoretic foundation.

Another advantage claimed is that CTTs provide a natural environment for program verification and synthesis, since CTTs have a built-in programming language and a program synthesis paradigm. However, the built-in languages are rather simple purely-functional languages. CTTs have no advantages when it comes to dealing with more sophisticated languages with imperative, concurrent or parallel features. Analogous synthesis paradigms can be explored in a classical setting if for example programs are represented syntactically and logic variables are used to stand in for program sections that remain to be synthesized. With such an approach both the programming language and the synthesis mechanism would be more open to exploration since neither would be be hard wired.

4 Conclusions

- *No*, the fundamental tension between constructive type theorists and classical mathematicians is not resolvable: the mathematics that each are interested in is just too different.

- *Yes*, many aspects of the tension between constructive type theorists and classical mathematicians are resolvable: in particular, there do seem to be a number of opportunities for productive collaboration, especially in the engineering of interactive theorem-proving systems.

5 Discussion

Here I've reconstituted a few of the comments made at the end of the talk from some rather sketchy notes that I took down. Hopefully no one's views are misrepresented. The comments are being checked with their ascribed authors at the moment.

- (*Holmes*) "One of the major features of CTTs is the role of proof objects". In proof theory, these proof objects can be more compact and convenient to work with than proof trees.

- (*Kapur*) "Present day constructivist often cite 19th century mathematics as being in their tradition, but this mathematics also fits in perfectly well with classical mathematics". Kapur also emphasized the size of the CTT community and expressed the need to have them involved in any QED venture.

- (*Trybulec*) "Too much of the QED manifesto was devoted to constructive concerns. There *is* a koine on which nearly all mathematicians agree. The QED project should try to take the simplest possible approach. The problems are hard enough as it is".

- (*McCarthy*) "Hilbert complained about being driven out of Cantor's paradise. Well, set theories like ZF allow us to be driven out the minimal amount".

- (*Trybulec*) "A development of Heyting Algebras was carried out in the classical system Mizar". Trybulec showed this to a colleague who thought the development contained a new result.

References

[AGNvS94] Thorsten Altenkirch, Veronica Gaspes, Bengt Nordström, and Björn von Sydow. *A User's Guide to ALF*. Chalmers University of Technology, Sweden, May 1994. Available on the WWW `file://ftp.cs.chalmers.se/pub/users/alti/alf.ps.Z`.

[BB85] Errett Bishop and Douglas Bridges. *Constructive Analysis*. Springer-Verlag, 1985.

[C+86] Robert Constable et al. *Implementing Mathematics with The Nuprl Development System*. Prentice-Hall, NJ, 1986.

[CCF+95] C. Cornes, J. Courant, J. Filliatre, G. Huet, P. Manoury, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The Coq proof assistant reference manual: Version 5.10. Technical Report 0177, INRIA, July 1995. Available from `ftp.inria.fr`.

[dB80] N. G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.

[For93] Max B. Forester. Formalizing constructive real analysis. Technical Report TR93-1382, Computer Science Dept., Cornell University, Ithaca, NY, 1993.

[How91] Douglas J. Howe. On computational open-endedness in Martin-Löf's type theory. In *Proceedings of Sixth Symposium on Logic in Computer Science*, pages 162–172. IEEE Computer Society, 1991.

[Jac95] Paul B. Jackson. *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*. PhD thesis, Cornell University, January 1995. Available as Cornell Computer Science Technical Report TR95-1509 from `http://cs-tr.cs.cornell.edu`.

[LP92] Zhaohui Luo and Robert Pollack. Lego proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, University of Edinburgh, May 1992. Available from `http://www.dcs.ed.ac.uk/lfcsreps/`.

[MRR88] Ray Mines, Fred Richman, and Wim Ruitenburg. *A Course in constructive Algebra*. Universitext. Springer-Verlag, 1988.

[Rud92] P. Rudnicki. An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*, Bastad, 1992. Chalmers University of Technology.

The Organization of a Data Base of Mathematical Knowledge

Martin Strecker[1]
University of Ulm, Germany

The talk started with a survey of the main difficulties the QED project has to face. These are, among others:

- Different foundational formalisms
- Different concepts of validity of theorems
- Consequently, different calculi yielding different proofs

Two different approaches can be envisaged for solving these difficulties. On the one hand, one can define a "root logic" serving as a meta-language for expressing and communicating statements and proofs developed in the particular object logics. Such an approach was judged to be a rather far-reaching goal not attainable in the near future.

On the other hand, a database of formalized mathematics and computer science can be constructed, starting from the large corpus of theories developed for some of the current systems. Apart from demonstrating the utility and applicability of formalized mathematics to "the public", such a database could improve the exchange of results within the QED community. For meeting this goal, the database would have to include a detailed description of objects, such as:

- Terminology, definitions
- Axiomatizations
- Mathematical theories
- Theorems
- Proofs and proof methods / tactics

A theory could for example be indexed by the following information:

- On which other theories does it depend?
- Which theories does it extend?
- Are there theory interpretations to other theories?
- Over which theories is it parameterized?

Apart from that, some meta-information would have to be added in order to account for different philosophies of mathematics. Such information might detail whether a proof uses a constructive or non-constructive argument or whether it is based on a contested axiom.

The discussion following the talk centered on the problem of retrieving information from a database. Since most information is currently stored in the form of text files, it is difficult to

[1]e-mail: `strecker@informatik.uni-ulm.de`

organize the data in such a way that they are amenable to “semantic” queries as opposed to a purely textual search. In particular, experience shows that finding previously proved theorems is a serious problem. It seems that so far no adequate solutions for these problems are known and further research on this topic is necessary.

Another issue addressed during this discussion (and resumed in several other occasions) was the question of whether QED should get involved in foundational debates, for example when deciding on the structure and contents of a database, or whether QED had not better develop practical methods for making existing formalizations accessible and for combining them. This could eventually lead to a methodology of problem solving similar to the approach advocated in software engineering: Decomposing a given problem into subproblems, until solutions can be found by the aid of powerful libraries.

One of the tangible results of the discussion was the proposal to provide easy access to the systems developed in the QED context via WWW. Links to systems could be kept on a centralized WWW page. By activating a link, the user could start a session with the respective system and either initiate proof tasks or at least submit queries about certain theories. It was mentioned that some systems (Nuprl, IMPS) already provide some of these services or will soon make them available.

Indefiniteness

Randall Holmes[1]
Math. Dept., Boise State Univ.

Abstract

The general topic of *undefined terms* was discussed and basic approaches were outlined.

1 Approaches

1.1 History

Russell's theory of descriptions, formulated in response to the views of Meinong, stands at the beginning of the topic.

1.2 Alternate Logics

Systems of free logic with an existence predicate (described in a preprint *Undefinedness* of Feferman) implement the logic of Russell's description operator, or, alternatively, can implement the approach of Meinong which Russell was trying to discredit. In either approach, the range of quantifiers is restricted to the objects which exist; in a "Russellian" approach free variables also range only over existing objects, whereas in a "Meinongian" approach free variables range over all objects, existent or otherwise.

These logics are good at handling partial functions. The PF logic of the IMPS theorem prover is a (Russellian) logic of this kind.

The common characteristic of these logics is that rules for well-formedness of terms can be kept simple, but well-formed terms do not necessarily denote, and an existence predicate is available to capture this information.

1.3 Typing Approaches

In this family of approaches, one adopts a type system designed to keep *undefined terms* ill-formed. This approach appears to require subtyping (for example, if we are defining division on the reals, we need the type of nonzero reals for the typing conditions). This would appear to lead to complex, even undecidable type schemes; this might not be a problem if one was already committed (as in some constructive type theories) to a very complex typing scheme.

The PF logic of IMPS is not a scheme of this kind, though it is the logic of a type theory; in PF logic, one has well-formed terms which do not denote and an existence predicate (this distinction was brought out in the discussion).

[1]e-mail: `holmes@math.idbsu.edu`

1.4 Default value approaches

In these approaches, well-formedness of terms is kept simple and all terms are treated as denoting. Terms which would normally be regarded as undefined are assigned "default values". A spectrum of approaches exists, ranging from assigning the same default value to every *undefined term* (or the same default value to every undefined term of each individual type) to an approach in which one carefully avoids providing any information about default values except that they exist. One tries to choose default values so as to keep theorems simple, but also keep them familiar-looking; it is probably a good idea to avoid having too many surprising theorems resulting from one's default convention. An example given by Boyer is a convention used by Morse in a treatment of von Neumann-Gödel-Bernays set theory (with proper classes) in which the universal class (which is a first-class object but not an element of any set) is used as the default value in all cases; this works surprisingly well in many cases.

These approaches do not interact well with the use of pointwise operations on partial functions (for example, consider the sum of the functions $x! - > 1/x$ and the constant 1 on the one hand and the function $x! - > (1/x) + 1$ on the other; suppose that *undefined values* are set to 0; these two functions have different values at 0 and so are distinct). This suggests the use of a special error value for undefined values of functions and restricting one's attention to "strict" functions (those which send the error value to the error value); such an approach would culminate in something like the semantics of Scott's models of the lambda-calculus. Notice that Morse's default value approach to set theory described above also handles this correctly.

2 Relation to QED

Different approaches to undefined terms create an obstruction to exchange of information between theorem provers which has nothing to do with underlying mathematical issues (there are likely to be other such obstructions arising from differences of mathematical convention rather than content).

The character of different provers may dictate which approach is most convenient: for example, in the prover which Holmes is working on, an equational prover without a built-in type system or an easy way to express side conditions, a default value approach was most natural. Thus, a commitment to a single approach throughout QED seems undesirable.

Holmes suggested the design of a menu of standard approaches to undefined terms and a set of protocols for translation between these approaches. An example: translation from a type-based approach to an existence predicate based approach should be easy.

Possible Use of Already Formalized Mathematical Knowledge

Manfred Kerber[1]
Fachbereich Informatik, Universität des Saarlandes,
Im Stadtwald 15, D-66041 Saarbrücken, Germany

1 Motivation

In this sessions the problems with a fixed representation language for QED have been discussed. In particular the objections of Gérard Huet against the feasibility of QED presented in the QED discussion at CADE-12 (1994) were taken as starting point. Huet said that there will be never any consent on the logical framework (type theory, set theory, constructive logic, classical logic, generic logic, ...). If this is true, QED must provide different formal systems for the representation of mathematical knowledge.

It seems to be quite obvious that different formalizations have their own advantages and drawbacks. Even for one single problem it might be that different formulations are appropriate, for instance, an explicit one in which a user can easily represent and recognize a theorem, and a more implicit one which is more suitable for a fully mechanical or a machine-supported proof. The advantages of different representations can easily be seen by means of sorts. In many situations a sorted formulation is more adequate for the formalization and the proof process (automatic as well as interactive) than an unsorted one. In some cases, however, it is not possible to use a (standard) sorted formalization, for instance, when you want to make a statement that a certain term has *not* a particular sort. The same situation holds for type theory compared to set theory.

Another important aspect concerns the reuse of big amounts of mathematical knowledge that has been accumulated in systems like MIZAR or NUPRL and should be reusable in QED.

2 Approach

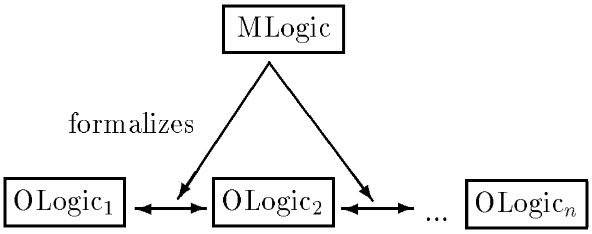
As a consequence of the problems with one fixed language, QED should not only support one single formal language, in which all statements have to be made, but a (not too big) variety of languages including the standard approaches (like set theory or type theory). In order to be able to transfer proofs from one format to another, it is necessary to have an exchange format for different syntactic objects, namely terms, formulae, assertions (axioms, definitions, theorems with their proof status), and proofs. This exchange format should satisfy in particular the requirements that it is easy to parse (prefix notation). Therefore it should be different from a user-friendly high-level format that can easily be read by humans.

In order to avoid a variety of unrelated languages, they must have provable relationships. Ideally there is one *constructive meta-logic*, for instance, the NUPRL-logic[2], in which the re-

[1]e-mail: `kerber@cs.uni-sb.de`, tel.: (+49) 681-302-4628.
An extended version of this abstract can be found in URL:
`http://js-sfbsun.cs.uni-sb.de/pub/papers/abstracts.html#Kerber95-QED`

[2]see: Robert L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

lationships between different formalizations can be formally proved. The main advantage of a constructive meta-logic consists in the possibility to extract from each meta-level proof an algorithm, which translates proof from one system to another.



While a classical meta-language can be used for the purpose of stating the relationship between different object logics as well, we propose to use a *constructive* meta-language. This has the advantage that by the constructive meta-proofs idealists can directly employ the theorems of another theory just by translating the theorem in their own language (and thereby avoiding the reconstruction of possibly lengthy proofs), while nominalists[3] can translate the theorems *and* the proofs in their own formalism. The proposed framework allows easily to integrate existing systems in the QED system.

3 Discussion

In the discussion the following points were made:

- There should be one common meta-logic only. Is PRA the best choice for a meta-logic?
- After the discussion of the proposal that there should be only one single object logic, the idea arose that there could be a tower of object languages such that each can be encoded in a lower level one, with some set theory as the least level one.
- In oder to restrict the number of object logics to a small number certain types of languages have to be offered like set theory and type theory.
- The meta-logic should include a theory of definitions and expansion of definitions.
- An alternative to a meta-logic would be the approach of having a collection of (program-level) translations between different formalizations and proof checking on this level.
- As the last point the question was discussed on which level proofs should be communicated. The general opinion was that as much information as possible should be contained in the proofs.

[3] For the notions idealist and nominalist in the context of automated theorem proving see: Francis Jeffry Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufmann.

Reflection; Practical Necessity or Not?

John Harrison[1]

Abstract

This paper summarizes my talk and the following discussions at the second QED workshop in Warsaw on 20–22 July 1995.

1 Are fully expanded proofs feasible?

Is it acceptable to reduce all mathematical proofs to formal constructions in something like a fairly orthodox Natural Deduction logical system? I believe that for the overwhelming majority of mathematics, the answer is 'yes'. We can split acceptability into two criteria: whether it is feasible to do it at all, and whether the user finds it congenial.

As for the user finding proofs congenial, there are well-established ways of programming higher level derived rules which ultimately decompose to the standard primitives, so only the computer need be concerned with that level of detail. Theorem provers in the LCF family implement this technique very effectively. In a QED-style undertaking, some such mechanism will be required anyway to automate various (domain-specific) inference patterns, and experience with HOL points to its great power and flexibility.

As for feasibility, it is widely accepted that almost all mathematics can *in principle* be reduced in this way. Of course, Gödel's theorems show that any given formal system is incomplete, in that there are true sentences unprovable in it. However such sentences are invariably theoretical pathologies with no obvious relationship to mainstream mathematics. The real question is how big the gap is between 'in principle' and 'in practice'.

We cannot answer this question conclusively, but we can point to some plausible intuitive evidence. Natural deduction systems are so-called because they really correspond closely to how mathematicians actually prove theorems. In practice, they use a few patterns of inference which constantly recur. There seems no reason to suppose that formalization will increase the size of proofs by more than a modest factor. This is supported by practical experience: the Mizar system has been used to formalize a wide variety of mathematics, and the proofs never become unmanageably large. The same experience is derived from the more modest mathematical formalizations undertaken in systems like HOL. Can anyone point to a mainstream mathematics text (for example one of the volumes of Bourbaki) and find an exception?

It is easy to construct true sentences which are provable in a given formal system but only with an unfeasibly long proof. But once again, it seems that like the Gödel sentence these are merely theoretical pathologies. And many proofs are hard to find, but this makes no difference to the business of actually formalizing them.

But even if the above argument is correct, there are accepted theorems whose proofs have never been written out in the conventional way. These are usually theorems which have been

[1]e-mail: `John.Harrison@cl.cam.ac.uk`

established with the aid of extensive computer checking of many cases; the four-colour theorem being the obvious example. (This point was made by Bill McCune.) It may be that here, at least close to the frontiers, it will not be feasible to produce a formal proof in the conventional sense. This situation seems more likely in theorems which are not really mainstream mathematics, but rather arise from verification work.

There are some technical arguments which suggest that any functional program which does some kind of proof can be internalized inside a formal system, and its effect simulated there. This is likely to induce a very substantial slowdown (probably greater if the program used imperative features like arrays and assignments), but only in extreme cases likely to make things completely impossible.

And there is considerable intellectual benefit for a foundational project like QED to insist of fully expanded proofs. It establishes a simple, canonical standard, and facilitates sharing between different systems. Even if this does hobble our ability to encompass some computer-checked proofs, this might be felt to be an acceptable price, especially as such theorems are usually rather rococo and peripheral to the main body of mathematics.

2 What if they aren't?

If fully-expanded proofs are not sufficient, what is to be done? One of the touchstones of the QED project is reliability. We can hardly just accept the results of ad-hoc checking programs written in various different computer languages. It seems that the only principled answer is something which is usually called *reflection*.

The basic idea is to verify the correctness of the computer program which 'proves' the theorem. That is, the semantics of the programming language concerned (C, LISP, FORTRAN or whatever) is formalized inside the system. Then it is verified that, on the basis of this semantics, a particular result of running a program (e.g. that "main" returns 0) indicates that some fact is indeed true (which in practice means 'provable', though as John McCarthy pointed out, one can quite well imagine taking the opportunity to move beyond the constraints of the particular formal system at the same time). This code is then run, or even incorporated into the system itself.

In practice there are difficulties in carrying out such a project. The real semantics of programming languages like C are very complicated. For ease of semantic description, still more actual verification, it's likely that one would have to program in quite a restricted subset. Every simplification and abstraction (e.g. ignoring numeric overflow) makes the result less and less reliable. Then there is the worry whether the intended semantics is correctly implemented by whatever machine/OS/compiler combination is used to run the code. Of course if the same system is used to run the theorem prover, that is already taken for granted. But by relying on reflection, the abstract description of what counts as validity is no longer describable in a couple of pages, but depends intimately on a colossally complex hardware/software system.

3 Logical reflection principles

Without the dangerous jump to verifying code then running it on real machines, one can still use some notion of reflection. In fact, when logicians talk about (logical) reflection they usually mean something like the following. We carry out a Gödel-style formalization of the logic's syntax and proof system inside itself. Then we augment the system with a new rule allowing $\vdash \phi$ to be deduced from $\vdash Provable(n)$ where n is the Gödel number of ϕ. This allows us to use certain kinds of meta-reasoning to establish that something is provable without actually constructing a proof.

This form of reflection is obviously much safer (actually the new reflection rule is a conservative extension provided the original system is 1-consistent, a pretty weak requirement). However it is also less obviously useful. Examples in the literature are for trivialities like using multiset equality to justify associative-commutative rearrangements, completely ignoring the question of whether multiset equality is any easier to prove. And in any case, a decent higher order logic or set theory can generally do all this ostensibly 'syntactic' reasoning without any extension of the logic (this is often referred to, following the work of Howe, as 'partial reflection').

4 Summary

There is little evidence that reflection principles are necessary for the vast body of mathematics. And the leap in conceptual complexity that it entails is best avoided except in the teeth of compulsion. There will be plenty of problems in formalizing mathematics, and the feasibility of expanded proofs does not yet seem one of the more important.

For a more detailed discussion of reflection principles, fuller presentations of the arguments adumbrated here, and an extensive bibliography, see the author's: 'Metatheory and Reflection in Theorem Proving: A Survey and Critique', Technical Report CRC-053, SRI International Cambridge Research Centre, also available on the Web as:

`http://www.cl.cam.ac.uk/users/jrh/papers/reflect.dvi.gz` .

Development of Analysis in the QED Project

F. Javier Thayer[1]

1 Introduction

The QED project is a cooperative effort in the area of automated mathematical reasoning. Many goals have been suggested for this common effort. A common element in most of the suggested goals is the construction of a rigorous mathematics database. By rigorous I mean that each entry in the database is a valid theorem stated in some formal theory.

Following is a summary of my presentation on Formalized Analysis at the 2nd QED workshop held in Warsaw, Poland July 20 to 22. My focus in the discussion is the styles of mathematical development appropriate for formalizing analysis in the QED project. I believe the answer to this question is time dependent.

2 Time Scales

In discussing the QED project and its goals, I suggest we establish some time scales. My subjective values for these time horizons are short term (6 months to 2 years), medium term (2 to 5 years), long term (5 to 15) years and the distant future 15 years or more. Though QED is clearly a project for the long term or maybe even the distant future, setting achievable short and medium term goals is crucial for the project for a number of reasons. Shorter-term goals provide:

- Feedback for testing ideas.
- Motivation for developers.
- Reassurance for funders.
- Useful applications, in education, symbolic mathematics and formal methods.

3 Approaches to Analysis

The terms "Big Theory" and "Little Theory" have been coined by Farmer, Guttman and Thayer to describe certain styles of mathematical development. I will only mention here some characteristics of each of these styles. For the Big Theory style there is usually a well understood formalism in which all reasoning is imbedded. Moreover, no special logical artifices (such as theory interpretations) are needed to apply theorems about structures since structures are first-class objects. I am using structure here in the sense of a structure such as a vector space, or topological space. For the Little Theories style one considers a large number of "small" axiomatic theories interrelated by theory interpretations. Structures such as vector spaces and so on are dealt with as axiomatic theories. One considers one instance or at best a small number of instances of such structures.

[1] e-mail: jt@linus.mitre.org

4 Big Theory: Advantages and Disadvantages

I would expect any development style using the Big Theory approach to be based on some form of set theory. This is an advantage, since foundational prerequisites for the working mathematician are almost zero. Regarding the Bourbaki approach as the prototypical "big theory" style, its is clear that the Big Theory style is conceptually elegant, extremely refined, general and highly traditional.

The biggest advantage of the Big Theory approach, is that quantification over theories is straightforward. This provides a whole range of ideas and techniques not easily available in the little theories style. For instance, topological methods can be applied to families of structures, to build and study structures such as fibre bundles, sheaves, and deformations. These ideas are an essential part of the analyst's toolkit.

Though the Big Theory approach allows for enormous flexibility, and the application of a whole range of techniques (as seen above), it has various drawbacks. One minor drawback, is that "big theory" has to be developed to the point where one can begin to define structures either as tuples or by some other artifice. Another drawback, is that the mathematical universe is too homogeneous. This makes pattern matching, and consequently application of results considerably more difficult. One way of dealing with this is to superimpose a sorting structure on the universe to facilitate pattern matching.

5 Little Theories: Advantages and Disadvantages

Little Theories provide pedagogically compact, often insightful approaches to various areas in mathematics. Basically, write down the axioms and you're in business. This makes short or medium term goals attainable. Similarly, much of analysis that only involves reasoning with inequalities, fits nicely into little theories since we are not using more difficult topological properties of mappings. Finally, the Little Theories approach fits in nicely with Type Theory. This provides syntactic cues for matching and theorem application.

On the negative side, Type Theory is inflexible. For instance, completions (algebraic and completions in particular) and other extensions (such as adjunction of points at infinity to a topological space) are very hard to deal with. Another difficulty of Little Theories is the need for special artifices to compensate for lack of machinery. For instance, functorial concepts cannot be easily accommodated within the little theories approach, so that topological arguments based on homology cannot be used.

6 Discussion

The presentation was followed by a discussion that focused on a number of issues including the suitability of Type Theory for mechanized mathematics. Though my presentation was not directly focused on types, there was some concern that by favoring the Big Theory approach, I argued against types. Although I do not consider types (or sorts) to be useful in the long term, I believe there use is an extremely helpful adjunct to the Little Theories style which in the shorter term is extremely valuable.

7 Summary

The Little Theories approach is pedagogically extremely useful, and in the short and medium term is capable of providing a framework for development of automated mathematics. It can work well with existing symbolic mathematics systems providing payoff in education and possibly other areas. As a long term goal, however, I believe that QED must include in its mathematical database, general theorems on topology and geometry, theorems about families of structures, theorems about functors such as enveloping algebras and so on. This is a powerful argument that some big theory approach to QED is required.

Mathematical Synthesis

Peter White[1]
Motorola Corp., USA

At the QED '95 conference in Warsaw, I presented the work we are doing here at Motorola to synthesize software and hardware from mathematical specifications. We are in the process of evaluating the Specware toolset, which is currently under development at the Kestrel Institute in Palo Alto, California. The Specware system has the following components:

1) An algebraic specification language, strongly typed, including subsorts and quotient sorts.

2) Specification morphisms, to describe the relationship between specifications.

3) Category theoretic operations such as colimit to combine specifications, providing for features such as parameters to specifications.

4) Sheaf / category theoretics refinement operations of specifications, allowing successive introduction of details. For example, a queue specification could be refined to a partial map, which could be refined to some list implementation for LISP code generation.

5) A theorem prover to discharge the proof obligations generated during the specification and refinement phases.

6) A graphical user interface, so that specifications can be manipulated using pretty pictures.

We are having success with describing systems using Specware, but the code generations proves to be arduous.

I thought I was able to contribute the perspective of *a-user-of* the theorem provers to the conference. From the perspective of a user, I think that features that make life easier (such as types and heavy duty set theory) are necessary, mathematicians will not use a system that is hard to use. Eventually, the system will require an interface that allows the notation to be as free wheeling as that found in textbooks. Perhaps an acceptance test would be to see if someone doing tensor analysis, complete with all of the tricks they play on superscripts and subscripts, would like to check his work with the system. Tensor analysis might also be a good domain since it is so easy to make an error with all of those tiny super/sub scripts.

I think another theme of the conference is to try and find a way to unify the different logics and theorem provers, in such a way that everyone is working to the same end, the creation of a large database of mathematical knowledge that is useful to industry and to mathematicians.

[1]e-mail: `peter@opus.geg.mot.com`

The Mutilated Checkerboard in Set Theory

John McCarthy[1]
Computer Science Department,
Stanford University

An 8 by 8 checkerboard with two diagonally opposite squares removed cannot be covered by dominoes each of which covers two rectilinearly adjacent squares. We present a set theory description of the proposition and an informal proof that the covering is impossible. While no present system that I know of will accept either the formal description or the proof, I claim that both should be admitted in any *heavy duty set theory*.[2]

We have the definitions

$$Board = Z8 \times Z8, \tag{1}$$

$$mutilated\text{-}board = Board - \{(0,0),(7,7)\}, \tag{2}$$

$$\begin{aligned} &domino\text{-}on\text{-}board(x) \equiv (x \subset Board) \wedge card(x) = 2 \\ &\wedge (\forall x1\ x2)(x1 \neq x2 \wedge x1 \in x \wedge x2 \in x \\ &\supset adjacent(x1,x2)) \end{aligned} \tag{3}$$

and

$$\begin{aligned} &adjacent(x1,x2) \equiv |c(x1,1) - c(x2,1)| = 1 \\ &\wedge c(x1,2) = c(x2,2) \\ &\vee |c(x1,2) - c(x2,2)| = 1 \wedge c(x1,1) = c(x2,1). \end{aligned} \tag{4}$$

If we are willing to be slightly tricky, we can write more compactly

$$adjacent(x1,x2) \equiv |c(x1,1) - c(x2,1)| + |c(x1,2) - c(x2,2)| = 1, \tag{5}$$

but then the proof might not be so obvious to the program.

Next we have

$$\begin{aligned} &partial\text{-}covering(z) \\ &\equiv (\forall x)(x \in z \supset domino\text{-}on\text{-}board(x)) \\ &\wedge (\forall x\ y)(x \in z \wedge y \in z \supset x = y \vee x \cap y = \{\}) \end{aligned} \tag{6}$$

Theorem:

$$\neg(\exists z)(partial\text{-}covering(z) \wedge \bigcup z = mutilated\text{-}board) \tag{7}$$

Proof:

We define

$$x \in Board \supset color(x) = rem(c(x,1) + c(x,2), 2) \tag{8}$$

---

[1] e-mail: `jmc@cs.stanford.edu`,   `http://www-formal.stanford.edu/jmc/`

[2] The Mizar theorem prover essentially accepts the definitions, but seems to be far from providing a proof.

$$domino\text{-}on\text{-}board(x) \supset (\exists u\ v)(u \in x \wedge v \in x \wedge color(u) = 0 \wedge color(v) = 1), \tag{9}$$

$$partial\text{-}covering(z) \supset card(\{u \in \bigcup z | color(u) = 0\}) = card(\{u \in \bigcup z | color(u) = 1\}), \tag{10}$$

$$card(\{u \in mutilated\text{-}board | color(u) = 0\}) \neq card(\{u \in mutilated\text{-}board | color(u) = 1\}), \tag{11}$$

and finally

$$\neg(\exists z)(partial\text{-}covering(z) \quad \wedge \quad mutilated\text{-}board = \bigcup z) \tag{12}$$

Q.E.D.

To Type or Not To Type

Piotr Rudnicki[1]
Department of Computing Science, The University of Alberta,
Edmonton, Alberta, Canada T6J 2H1

1 Introduction

N. G. de Bruijn in his letter to R. S. Boyer of August 19, 1994 which appeared at the QED mailing list wrote:

> ... in type systems like AUTOMATH the notion of elementhood can be seen as typing. Customers who like typed sets (let me call these customers TSC's) introduce a real variable by a single block opener, like
>
> `let x be real number`
>
> where `real number` is taken as a type. So for the TSC's it is the introduction of a typed variable. The customers who prefer to think in terms of untyped sets (let me call them USC's), however, have to order by means of two block openers:
>
> `let x be a set`
>
> and
>
> `assume this x is a real number.`
>
> In their case we have to realize that `x is a real number` is the property of the set `x`. The first one of the two block openers is the introduction of a typed variable (where the type is `set`), the second one is the assumption.

The word *type* is used with a number of different meanings and someone noticed that for the use of *type* as above, probably one should use *sort*. In MIZAR this notion is named *syntactic type*.

The question that we would like to address is: *What are the advantages and disadvantages of using syntactic types?*

In MIZAR one can work without syntactic types (types for short). However, the MIZAR data base has been developed using a hierarchy of types as it seems closer to everyday mathematical practice and the developers of MIZAR articles usually followed that practice as they knew it.

2 What are syntactic types in MIZAR?

Let us look at an example of how the type of *finite sequence of X* is defined. In article `FINSEQ_1` we have the following definition:

[1]Supported in part by NSERC Grant No. OGP9207, e-mail: `piotr@cs.ualberta.ca`

```
definition let D be set;
 mode FinSequence of D -> FinSequence means
  rng it c= D ;
 existence  proof
    :: Here we have to demonstrate that there is an object of type
    :: FinSequence whose range is a subset of D
 end;
end;
```

FinSequence of is a type constructor which given a specific set as a parameter results in a type expression which can be used then to type objects. Later on we can then write:

```
let FST be FinSequence of NAT;
```

The denotation of FinSequence of NAT is the set of all finite sequences whose range is a subset of NAT. There is a substantial difference between the above introduction of x and the following:

```
let FSU be set;
assume FSU is FinSequence of NAT;
```

which can be characteristic of the USC at the de Bruijn's restaurant. In MIZAR, the typing expressed at the time of introducing an object (let ...) is processed differently then the typing expressed in an assumption. The type information contained in the object declaration is always automatically taken into account by the MIZAR semantic analyzer and inference checker, while the information contained in an assumption must be explicitly referenced and is effective only for the inference checker.

In the MIZAR abstract of article FINSEQ_1 the definition of FinSequence of ... appears as:

```
 definition let D be set;
  mode FinSequence of D -> FinSequence means
:: FINSEQ_1: def 4
 rng it c= D ;
 end;
```

and the label FINSEQ_1:def 4 is used to make a reference to the definiens of FinSequence of D when needed. (The proofs do not appear in MIZAR abstracts.) The above definition says that FinSequence of D is a FinSequence with an additional condition given by the definiens. FinSequence is the mother type of FinSequence of D and FinSequence of D is a subtype of FinSequence. Let us have a look at the definition of FinSequence (in its abstract format, without proof of existence):

```
definition
  mode FinSequence is FinSequence-like Function;
end;
```

FinSequence turns out to be a shorthand for a Function with the attribute FinSequence-like.

This attribute, applicable to Function is defined as follows:

```
 definition
  mode FinSequence-like -> Function means
:: FINSEQ_1: def 2
 ex n st dom it = Seg n ;
 end;
```

The above is read: the domain of a `FinSequence-like Function` is equal to an initial segment of natural numbers. And below we quote the sequence of definitions that lead from the primitive notion of `set` to the notion of `Function`, again we skip the correctness conditions:

```
definition
 mode Relation-like -> set means
:: RELAT_1: def 1
  x ∈ it implies ex y,z st x = [y,z];
end;

definition
 mode Relation is Relation-like set;
end;

definition let X be set;
 pred X is Function-like means
:: FUNCT_1: def 1
 for x,y1,y2 st [x,y1] ∈ X & [x,y2] ∈ X holds y1 = y2;
end;

definition
 cluster Relation-like Function-like set;
end;

definition
 mode Function is Function-like Relation-like Any;
end;
```

The MIZAR semantic analyzer takes into account the entire chain of mother types. In particular, the object `FST` introduced above besides being a `FinSequence of D`, is also a `FinSequence`, a `Function`, a `Relation`, and of course a `set` as `set` is the root of the tree of types. All operations defined for a mother type are applicable also to the subtype. For instance, function application is defined as follows:

```
 definition let f be Function, x be Any;
  func f.x -> Any means
 ::  FUNCT_1:  def 4
 [x,it] ∈ f if x ∈ dom f otherwise it = 0;
end;
```

And therefore we can write `FST.10` which is always denoting. Note, that we must not write

FSU.10 as function application requires that the left argument of the dot is a function which does not follow from the declaration of FSU and no assumption regarding FSU is taken into account by the semantic analyzer.

Note that although FST.10 is denoting its type is Any, (Any and set are synonyms). Can we force FST.10 to be of type Nat? We found the definition of certain functor π in the MIZAR data base:

```
definition
  let X be set, D be non empty set, p be PartFunc of X,
      D, i be Any;
 assume  i ∈ dom p;
 func π(p,i) -> Element of D means
:: FINSEQ_4: def 4
   it = p.i;
end;
```

However, we must not write π(FST,10) yet, as FinSequence of NAT is not a partial function (PartFunc of NAT,NAT) in the MIZAR sense. In fact, PartFunc does not have Function in the chain of its mother types! The mode (type constructor) PartFunc has been derived as follows:

```
definition
 let X,Y;
 mode Relation of X,Y -> Subset of [:X,Y:] means
:: RELSET_1: def 1
  not contradiction;
end;

definition
 let X,Y;
 cluster -> Relation-like Subset of [:X,Y:];
end;

definition let X,Y;
  cluster Function-like Relation of X,Y;
 end;

definition let X,Y;
  mode PartFunc of X,Y is Function-like Relation of X,Y;
 end;
```

How can we make MIZAR treat a FinSequence of D as a PartFunc? For this we use a mechanism known as redefinition of modes. First, we show that the attribute FinSequence-like is applicable to partial functions (article FINSEQ_4):

```
definition let D be set;
 cluster FinSequence-like PartFunc of NAT,D:
```

```
  existence proof
    :: Here we have to show that there exists a partial function
    :: which is FinSequence-like.
end;
```

And now we can redefine `FinSequence of D` as a `PartFunc of NAT, D`.

```
definition let D be set;
 redefine mode FinSequence of D -> FinSequence-like PartFunc
     of NAT,D;
  coherence proof
    :: Here we have to show that every FinSequence of D is
    :: a PartFunc of NAT, D
end;
```

With this redefinition `FST` besides being whatever it was before is also a `PartFunc of NAT, NAT` and the expression π`(FST, 10)` is correctly typed.

3 Disadvantages of using syntactic types

- Syntactic types are an additional complication of any proof-checking system and as such decrease the reliability of the system and of its implementations.
- There is the temptation to define new notions on types that are too narrow.
- Type definitions need to be proven correct:
  - Since types have non-empty denotations we have to prove existence of an object of a newly defined type.
  - When we redefine the meaning of a type we have to prove coherence, namely that the type being redefined is coherent with the newly added mother type.
- The language must provide means for changing type of an object. In MIZAR this is done by the `reconsider` construction. For example:

  ```
  let x be Nat;
  . . .
  reconsider x as Integer by INT_1:9;
  ```

  and from now on `x` is treated as an `Integer`.

  Many a time the type change is performed by some artificially looking manipulations like introduction of otherwise not needed casting function or using existential quantifier and equality within a formula to “smuggle” information that an object of a type is equal to some object of a different type.

4 Advantages of using syntactic types

- A collection of some objects may be too big to be a set; however, we may introduce a type to name the collection. For example, we can define the type `Group` and objects of this type may have an arbitrary set as their carrier, while there is no possibility to define the set of all groups in MIZAR.

- The information carried by types of objects can be processed more efficiently by a specialized machine rather than the general inference engine.

  - Computing the possible types of an object and its attributes (subtypes) involves only sentential calculus.
  - The number of possible unifications when the types are taken into account is substantially reduced.

- Types facilitate overloading of symbols. For example, multiplication of real numbers is announced as (in `HIDDEN`):

```
definition let x,y be Element of REAL;
 func x · y -> Element of REAL;
  commutativity;
end;
```

and later on characterized axiomatically in `AXIOMS`. However, we can redefine this multiplication for natural numbers with a natural result. This is done in article `NAT_1` as follows:

```
definition let n,k;
redefine
 func n · k -> Nat ;
  coherence proof
    :: Here we have to prove that indeed when we multiply two
    :: natural numbers we obtain a natural number as a result.
  end;
end;
```

- Hidden arguments. Not all arguments of a function or predicate have to be explicitly mentioned—some may be inferred from types of explicit arguments. In `CAT_1` we find the following definition of the composition of morphisms in a category:

```
definition let C be Category;
           let a,b,c be Object of C;
           let f be Morphism of a, b;
           let g be Morphism of b, c;
 assume A: Hom(a,b)<>∅ & Hom(b,c)<>∅;
   func g·f -> Morphism of a,c means
    .  .  .  Definiens omitted
 existence proof Proof omitted end;
 uniqueness;
end;
```

The pattern of the defined operation has two explicit arguments, the morphisms in a category, and four implicit arguments that have to be reconstructed from the explicit arguments to fit what is stated in the definition. It is hard to imagine stating explicitly all six arguments when we use the composition of morphisms, which seems unavoidable if we do not use syntactic types.

- Reservations. With types, we can announce that certain identifiers are used to denote objects of specific types, and later when using the identifiers, we do not have to repeat the typing information. This feature of MIZAR saves a lot on writing, although some authors do not always use it.

Cooperation of Automated and Interactive Theorem Provers

Bernd, Ingo Dahn[1]
Humboldt-University, Institute of Pure Mathematics
Ziegelstr. 13a, D-10099 Berlin

QED is too big a task to be achieved by a single group. Therefore, it will demand the cooperation of different groups which have developed up to now their specific systems for their specific purposes. QED has to use the skills and achievements of these groups. Therefore, the introductory talk discussed possible ways of cooperation of existing and forthcoming theorem provers. There are several technical and social problems to overcome for such a cooperation. These were examined based on the experience of the ILF system within the research program *Schwerpunktprogramm Deduktion* of the Deutsche Forschungsgemeinschaft. Problems of parallelism in deductions have not been touched.

It is argued, that cooperation of provers gives stronger and more flexible systems. Within a cooperation the value of a prover will judged according to its ability to augment other provers.

We can distinguish theoretically the following four types of cooperation.

1 Monolithic

In this approach there is a single system combining the best properties of the subsystems. These subsystems cooperate by procedure calls. There seemed to be a consensus among the workshop participants that this approach is unrealistic.

2 Cooperation in Proof Production

In this approach, provers must be able to incorporate periodically new results produced by other systems during their work. They must output intermediate results that can be useful for others. Recently, within the German *Schwerpunktprogramm Deduktion*, Jörg Denzinger (`denzinge@informatik.uni-kl.de`) has argued that the adaptation of existing theorem provers to such a type of cooperation is a promising task. However, this requires a rather restrictive description of what should be communicated between the systems. Moreover, for existing theorem provers it is often impossible to incorporate more than literals during their work into the basic data structures.

3 Server for Proof Production

In this approach a general purpose - often interactive - system distributes subtasks to its subsystems. These try to solve subtasks without further cooperation. All implemented cooperating systems which are known to the author use this type of cooperation. The effect of this kind of cooperation depends on the distribution of subtasks. This is facilitated, if the specific strengths

[1]e-mail: `dahn@mathematik.hu-berlin.de`

of the integrated provers are known. Therefore, cooperative provers which take specific properties of a theory or domain into account are best suited here. It is also possible, to use prover competition within the system to determine the suitability of the provers for a given task.

4 Cooperation in Proof Exchange

In this approach, provers supply lemmas that can be loaded by other provers before starting a proof search. Here, the provers can run completely independent at different times and at different locations, connected by a network that gives access to a library of theorems. This is likely to be the most important form of cooperation within the QED project. It requires new tools for proof administration and exchange. It is desirable, to support the combination of proofs from different systems.

Experience in ILF

ILF is an interactive theorem prover which integrates the automated theorem provers Otter, Discount, Setheo and KoMeT. Its development is supported by the Deutsche Forschungsgemeinschaft. In ILF the user can communicate with the automated provers through a uniform graphical user interface. The integrated systems remain completely independent and no change in their code is required. Problems and effects of this cooperation within ILF and of the necessary cooperation with the authors of these systems have been discussed.

ILFs ProofPad

The ProofPad is a configuration of ILF where the user enters lines of a proof and the automated provers in the background try to prove, that each line is a logical consequence of the preceding lines. Therefore, the user does not have to know theorem provers or logical calculi. Experiments in ILF in the field of lattice ordered groups show, that by the use of automated provers it suffices, that the user enters about 10% of the proof interactively to have it formally verified, see a sample manuscript:

```
anonymous ftp from:    info.mathematik.hu-berlin.de ,   (141.20.54.8)
password:              (enter own identification)
in file:               /pub/ilf/busulini.dvi .
```

The Schwerpunktprogramm Deduktion

An important criterion for the projects within the *Schwerpunktprogramm Deduktion* is their ability to cooperate. Moreover the program has special funding for cooperation. All this turned out to be very helpful for the work of an integrating project like ILF. It should also give hints for establishing a cooperation of QED projects. Within ILF, tools have been developed that take demands of other groups in the Schwerpunktprogramm into account. The TreeViewer - a tool for the visualisation and manipulation of directed acyclic graphs - is a separate program that can also be used as part of the graphical user interface of other theorem provers. It can be obtained from:

```
anonymous ftp from:   info.mathematik.hu-berlin.de ,   (141.20.54.8)
password:             (enter own identification)
in file:              /pub/ilf/tview.tar.Z  .
```

The natural language proof presentation facilities of ILF have been developed to be largely independent of a specific calculus. They are made available to other groups by the automated ILF mail server that takes proofs and returns LaTeX sources. You can get more information on the ILF server by sending a mail with text help to `ilf-serv@mathematik.hu-berlin.de` .

Other Problems

All provers currently integrated into ILF prove theorems in fragments of first order logic. The cooperation could be extended provers for other logics if it is known whether provability in one of these logics implies provability in the other logics in use.

What are the Connections, Inter-relations and Antipathies Between Proof Checking and Automated Theorem Proving?

Deepak Kapur[1]

There are more commonalities and connections between automated theorem proving and proof checking than are often mentioned. Automatic theorem provers (e.g. *OTTER* and *RRL*) can prove many interesting, nontrivial theorem automatically. For most nontrivial theorems, however, even automatic theorem provers (ATPs) need guidance for finding their proofs. This guidance to ATPs could be in the form of setting knobs/parameters to select appropriate heuristics and inference methods, and/or providing intermediate lemmas that could potentially help in finding a proof. Because of this, automated theorem proving programs are also called *mechanical theorem provers*, *proof assistants*, etc. In this sense, they are similar to proof checkers (PCs).

ATPs and PCs differ in the degree of automation supported by them, and level of granularity of inference steps. As the name suggests, proof checkers are used to check manual proofs. At every proof step, a PC user may have to specify the next inference rule to be applied.

We highlight interrelationships between automated theorem proving and proof checking, discussing possibilities for cooperation between the developers of ATPs and PCs. We review different emphasis in concerns of two different strands of research in mechanical reasoning.

1 Finding Proofs

In an ATP, the emphasis is on finding a proof whereas in a PC, the emphasis shifts to proof checking. It is expected that a user has already done a rough detailed proof on paper before it approaches a PC. The user interacts with the PC primarily to find any gaps and/or errors in a manual rough proof. In contrast, the objective for using an ATP is to automatically generating a proof, or if that does not seem possible, to get as much help from the ATP in finding a proof. The user is not expected to have a proof already.

Because of different objectives, most PCs implement “small” inference steps, which are used by the user to check the manually obtained hand proof. In contrast, an ATP supports hard-wired large inference steps, typically implemented using smaller primitive inference steps which may or may not be made available to a user. Heuristics play a crucial role in the effectiveness of an ATP. There are many ways to combine primitive inference steps to be used for finding proofs in an ATP, consequently many parameters must be selected and set to an appropriate value. This makes ATPs quite flexible and effective in their use for experts, but difficult for new users.

1.1 Failed Proof Attempts

The mode of proof generation using a PC is quite different than using an ATP. Proof generation is done on a PC under user control, so if a proof attempt does not succeed, the user is likely to know where and why it is failing. This may point to a gap or an error in a manual proof, leading to the user rethinking about the manual proof.

[1]e-mail: kapur@cs.albany.edu

When a proof attempt does not succeed using an ATP, then it is often not easy for a novice user to figure out a reason why the proof attempt might not have succeeded. An expert user of an ATP would look at the transcript, guess whether a lemma is needed or a different proof strategy should be used. A proof is thus generated using "trial and debug" mode, much like a hand proof is obtained or a program is constructed. It is this aspect where there is considerable scope for research and for providing aids to users to help extract useful information from an unsuccessful proof attempt.

ATP builders would have to provide modes in which powerful inference methods supported by an ATP can be invoked under user control, much like a proof checker. Having an ATP run like a proof checker can be useful to attack interesting, nontrivial theorems, where user guidance is necessary. PC developers should not hesitate to adapt automated reasoning techniques commonly used in ATPs so that PC systems become less tedious and more effective to use.

2 Foundations of ATPs and PCs

PC builders have been very concerned about foundational issues related to proof generation and proof finding. Whereas interactive sequent based proof checkers support many inference steps (an inference step for introduction and elimination of every logical connective), a PC supports a small set of inference steps, but provides mechanisms for combining these inference steps to get larger derived inference steps. The main rationale is that soundness and related foundational issues can be more easily ascertained with a small set of primitive inference steps than with a large set of derived big inference steps. Soundness/correctness of combining mechanisms becomes, however, a major burden.

Soundness is a serious concern for ATPs which implement complex heuristics combining primitive inference steps. A small, trivial mistake can generate incorrect proofs thus shaking any confidence in an ATP; one starts wondering whether proofs obtained earlier are correct.

Developers of PCs have emphasized construction of detailed, fine-grained proof objects whereas proof objects are considered a burden by ATP developers, even sometimes a nuisance as generating them is not viewed as being of any use.

3 Cooperative Building of Knowledge Base

The QED project should build on the work of both the ATP and PC groups. One possible approach is to share proofs generated by different ATPs and PCs for building on each other's work. There is a need to develop mechanisms for sharing proofs and permanently storing them in a data base that theorem provers and proof checkers can interface with. A common notation (syntax) for proofs must be developed. How can an ATP or a PC use proofs generated by other ATPs/PCs in its work? This may require developing front-ends for ATPs and PCs in order to use proofs.

4 What is an Acceptable Proof?

Since the first QED workshop, there has been considerable interest among ATP developers to start thinking about proof objects underlying ATPs, how proof objects should be generated, and subsequently used. The Eves group of ORA, Canada, for instance, has successfully demonstrated how proof objects can be generated for Eves, a sophisticated system supporting reasoning by induction, employing decision procedures and other heuristics. McCune has developed a method for generating proof objects from Otter using basic inference steps of binary resolution, factoring on propositional calculus and instantiation. I outlined some preliminary ideas for possibly generating a proof object for a rewrite rule based prover such as *Rewrite Rule Laboratory* (*RRL*).

A consensus is emerging that a proof object should provide justifications for propositional reasoning (i.e., rules for boolean connectives), equality reasoning on ground terms (i.e., reflexivity, symmetry, transitivity, and functional application preserving equality), and universal instantiation. Perhaps some doubts can be raised about justification for the instantiation rule because of the discussion about indefiniteness earlier. There are issues related to instantiating a free variable with a term which has no value or no meaning. There are also issues related to typed variables. Can we instantiate a free typed variable if there is no guarantee that the type denotes an nonempty set? These concerns need to be adequately addressed as they are likely to affect the definition of acceptable proof objects.

There does not seem to be any agreement about a common meta logic regarding proofs by induction. One view is that set theory can serve as an appropriate meta logic, given that induction can be obtained as a derived rule in set theory, for example in ZFC or heavy-duty set theory proposed by McCarthy. This proposal is considered by others as one similar to writing programs in the language of Turing machines or in an assembly language. There is also Mizar view that first order set theory is adequate except that for developing proofs and formalization, it is convenient to have available a second-order feature which is really a syntactic mechanism to provide ability to substitute for meta variables over formulas.

Some of us from programming background believe that we need to agree on some mechanism for introducing structured objects. One proposal in that direction is to adopt Feferman's *Primitive Recursive Arithmetic* (*PRA*) that provides the notation of S-expressions (trees) as a basic data structure along with numbers. Others would like to support a mechanism for defining abstract data types generated by finite set of constructors (something similar to *shell* principle of Boyer and Moore's logic). More discussion is needed on this important issue as inductive reasoning may be critical in many QED applications.

4.1 Objects generated by decision procedures and other algorithms

Many ATPs implement decision procedures for commonly used theories so that a user of these systems is not burdened with having to prove simple obvious facts. Popular theories whose decision procedures are directly encoded into ATPs, are propositional calculus, theory of equality, and linear arithmetic (quantifier-free theory of numbers). Even developers of PCs have recognized the need for supporting such decision procedures, and there is an increasing trend towards providing specialized tactics implementing decision procedures for some theories.

Generating proof objects based on inference steps in a decision procedure is an interesting research issue. The Eves attempt is a good start.

As more capabilities are included in reasoning systems, particularly algorithms from symbolic computation (computer algebra) system, there would be a need to generate proof objects related to such computations also. It appears that for operations such as factorization of polynomials, computing integrals and differentiation, it is perhaps not that difficult to develop a proof object in the form of a certificate that ensures that the result is indeed correct, without having to know how the result is computed. However, this may not be straightforward in other nontrivial computations, such as resultants, Gröbner basis and other elimination techniques. There are new issues raised due to generation of huge objects and the so-called "intermediate expression swell" problem.

4.2 Representational Issues

There are other issued related to computer representation of a proof object. Should it be a linear object, much like a Hilbert style proof, or it should be a directed acyclic graph allowing sharing among parts? Should it be a hierarchical object or a nested object? What additional structure should a proof object have? Should it be a λ term?

The Mutilated Chessboard Problem - *checked by Mizar*

Grzegorz Bancerek[1]
Institute of Mathematics,
Polish Academy of Science, Warsaw

Abstract

The problem presented by John McCarthy during his lecture "Heavy duty set theory"[2] has been resolved here. The final version of that article has been made with the commitment of Andrzej Trybulec.

Proofs have been written in Mizar language and were checked by PC Mizar system. In this paper only definitions and theorems are published apart from proofs (this is the so called 'mizar abstract'). Then it has been automatically translated into English. The format of this article is exactly the same as in journal *Formalized Mathematics*.

MML Identifier: `M_BOARD`.

The articles [10], [12], [2], [13], [14], [5], [4], [6], [11], [9], [8], [7], [3], and [1] provide the notation and terminology for this paper.

We follow a convention: x, z will be sets, i, j, k will be natural numbers, and u, v will be elements of $[:\mathbf{N}, \mathbf{N}:]$.

One can prove the following proposition

(1) If $(i \bmod k) + (j \bmod k) < k$, then $(i + j) \bmod k = (i \bmod k) + (j \bmod k)$.

Set $\mathit{Board} = [:\mathrm{Seg}\, 8, \mathrm{Seg}\, 8:]$ and $\mathit{MBoard} = \mathit{Board} \setminus \{[1,1],[8,8]\}$.

Let x be an element of $[:\mathbf{N}, \mathbf{N}:]$. Then $x_{\mathbf{1}}$ and $x_{\mathbf{2}}$ are natural numbers.

Let x_1, x_2 be elements of $[:\mathbf{N}, \mathbf{N}:]$. We say that x_1 is adjacent to x_2 if and only if:

(Def.1) $|(x_1)_{\mathbf{1}} - (x_2)_{\mathbf{1}}| = 1$ and $(x_1)_{\mathbf{2}} = (x_2)_{\mathbf{2}}$ or $|(x_1)_{\mathbf{2}} - (x_2)_{\mathbf{2}}| = 1$ and $(x_1)_{\mathbf{1}} = (x_2)_{\mathbf{1}}$.

Let x be a set. We say that x is domino-on-board if and only if:

(Def.2) $x \subseteq \mathit{Board}$ and $\overline{\overline{x}} = 2$ and for all elements x_1, x_2 of $[:\mathbf{N}, \mathbf{N}:]$ such that $x_1 \neq x_2$ and $x_1 \in x$ and $x_2 \in x$ holds x_1 is adjacent to x_2.

Let z be a set. We say that z is partial-covering if and only if the conditions (Def.3) are satisfied.

[1]e-mail: `bancerek@impan.gov.pl`
[2]see: *The Mutilated Checkerboard in Set Theory* by John McCarthy, pp. 25 – 26.

(Def.3)(i) For every set x such that $x \in z$ holds x is domino-on-board, and

(ii) for all sets x, y such that $x \in z$ and $y \in z$ holds $x = y$ or $x \cap y = \emptyset$.

Let z be an element of $[:\mathbf{N}, \mathbf{N}:]$. The functor color($z$) yielding a natural number is defined by:

(Def.4) $\mathrm{color}(z) = (z_{\mathbf{1}} + z_{\mathbf{2}}) \bmod 2$.

The following three propositions are true:

(2) Let z be a set. Suppose z is partial-covering. Let x be a set. Suppose $x \in z$. Then there exist elements x_1, x_2 of $[:\mathbf{N}, \mathbf{N}:]$ such that $x = \{x_1, x_2\}$ and $\mathrm{color}(x_1) = 0$ and $\mathrm{color}(x_2) = 1$.

(3) For every set z such that z is partial-covering holds $\{u : u \in \bigcup z \ \wedge \ \mathrm{color}(u) = 0\} \approx \{v : v \in \bigcup z \ \wedge \ \mathrm{color}(v) = 1\}$.

(4) It is not true that there exists a set z such that z is partial-covering and $\bigcup z = \mathit{MBoard}$.

References

[1] Grzegorz Bancerek. Cardinal numbers. *Formalized Mathematics*, 1(**2**):377–382, 1990.

[2] Grzegorz Bancerek. The fundamental properties of natural numbers. *Formalized Mathematics*, 1(**1**):41–46, 1990.

[3] Grzegorz Bancerek. Zermelo theorem and axiom of choice. *Formalized Mathematics*, 1(**2**):265–267, 1990.

[4] Grzegorz Bancerek and Krzysztof Hryniewiecki. Segments of natural numbers and finite sequences. *Formalized Mathematics*, 1(**1**):107–114, 1990.

[5] Czesław Byliński. Functions and their basic properties. *Formalized Mathematics*, 1(**1**):55–65, 1990.

[6] Agata Darmochwał. Finite sets. *Formalized Mathematics*, 1(**1**):165–167, 1990.

[7] Takaya Nishiyama and Yasuho Mizuhara. Binary arithmetics. *Formalized Mathematics*, 4(**1**):83–86, 1993.

[8] Jan Popiołek. Some properties of functions modul and signum. *Formalized Mathematics*, 1(**2**):263–264, 1990.

[9] Andrzej Trybulec. Domains and their Cartesian products. *Formalized Mathematics*, 1(**1**):115–122, 1990.

[10] Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990.

[11] Andrzej Trybulec. Tuples, projections and Cartesian products. *Formalized Mathematics*, 1(**1**):97–105, 1990.

[12] Michał J. Trybulec. Integers. *Formalized Mathematics*, 1(**3**):501–505, 1990.

[13] Zinaida Trybulec and Halina Święczkowska. Boolean properties of sets. *Formalized Mathematics*, 1(**1**):17–23, 1990.

[14] Edmund Woronowicz. Relations and their basic properties. *Formalized Mathematics*, 1(**1**):73–83, 1990.

What Can QED Offer to Mathematics?

Manfred Kerber[1]
Fachbereich Informatik, Universität des Saarlandes,
Im Stadtwald 15, D-66041 Saarbrücken, Germany

1 Correctness

The main argument why QED can be of great practical use is the ongoing striving for correctness in mathematics at a level as high as possible: a system like QED is the answer of our time to that striving. That this correctness problem is not sufficiently solved in traditional mathematics can be seen in the history of false theorems and false proofs.

False theorems: The perhaps best studied example of a theorem that had to be corrected again and again is Euler's polyhedron theorem. Imre Lakatos[2] has described the cycle of stating the theorem, proving it, and finding counterexamples, then going into the next loop with a refined version of the theorem. How can it be possible that there are counterexamples to a proved theorem? This can be due to inaccurate definitions, hidden assumptions, or invalid arguments. The main problem in the case of Euler's polyhedron theorem has been what to consider as a polyhedron. A system like QED, however, would not accept any inaccurate definitions, hidden assumptions, or invalid arguments, but it forces a human user to be precise.

False proofs: While the problem of false theorems is much more grave than that of false proofs, false proofs, which can be patched, occur much more often. This problem arises since mathematical proofs are normally not given on a formal logical level, but on an intuitive level where the ideas how to build a proof are conveyed rather than the proofs itself. On such a level it is possible that a proof is not understandable or that certain non-trivial gaps may occur (e.g. in the proof of Fermat's Last Theorem), for which it is not clear whether they can be closed or not.

QED can support the author, the reviewer, and the user of a theorem against false theorems as well as against false proofs. For each theorem must exist a machine-checked proof.

2 Proof Presentation

Proving mathematical theorems is a social process. The verification of mathematical arguments is a key activity in mathematics and hence very sensitive for the development of mathematics. Therefore it is not a good idea to try to replace this process by checking proofs in QED. On the contrary, QED should be considered as a supplement to the traditional process of proof checking and supporting this process. In particular QED should support different levels of the proof display (such as the logic level, assertional level, expert level with a user model, interactive proof display, or even multi-media facilities).

[1]e-mail: `kerber@cs.uni-sb.de`, tel.: (+49) 681-302-4628
[2]see: Imre Lakatos. *Proofs and Refutations.* Cambridge University Press, 1976.

3 Information Retrieval

While correctness and proof presentation may not be the main gains mathematicians can draw from QED, they may be very interested in using a large data base of definitions and theorems, when it is well-organized and easy to use. A large data base of mathematical facts in QED can easily provide useful information like the interdependence of certain axioms, definitions and theorem, which is hard to obtain without computer support.

4 Future Prospects

In order to come up with a system that finds a broad distribution, QED eventually has to enable a human user to develop a machine-checked proof at least as easily as to do that on paper. Therefore it is necessary to provide strong support in the proof search in more areas. In order to do that standard automated theorem proving techniques as well as more high-level oriented approaches like proof planning and analogy should be used.

But even when this goal is reached a lot of additional functionality can be integrated in QED, for instance support for some standard procedure of mathematics, like the approach of changing the definitions in order to get the right theorem.

5 Discussion

In the discussion the following points were made:

- Correctness may be too weak an argument for mathematicians when confronted with the QED-approach, since at first a lot of basic work has to be done before they can really start. Furthermore errors in papers are normally not too important. Therefore QED has to do a lot of preliminary work. In MIZAR, for instance, thirty master theses were written. This led not only to a profound amount of formal mathematical knowledge, but had also the interesting side effect that the relationship between a teacher and a student changed a lot: the teacher does not force the student to be precise, but the system does and the teacher has the role of assisting the student. In order to increase the acceptance of QED, it would be ideal to expose students to formal proving already in an early stage of their academic training.

- It would be fine to have a collection of well-documented examples for false theorems and false proofs.

- A main practical argument for mathematicians to use QED could be the speed of publications. If the article is formally written in QED, cumbersome proof reading can be done mechanically and the referees can concentrate on the relevance of the paper rather than on the correctness.

General Discussion: *Where do we go from here?*

Deepak Kapur[1]

In this last session of the workshop, the following topics were raised for discussions by the participants.

1. Do we need a common meta logic? If so, what should it be?
2. How can we build a data base of machine-checked proofs based on past accomplishments?
3. How can we monitor the progress of different theorem proving projects vis a vis QED?
4. How should the QED mailing list be administered?
5. When and where should the next meeting be? What form should it take?

Note: We briefly elaborate on these topics. Other participants are requested and encouraged to inform me about the salient points that I might have missed.

Randall Holmes announced that he had some ideas about a common meta logic for different theorem provers and proof checkers. He was planning to write a position paper based on those ideas for sharing with others. That paper would be available on Holmes' web page.

Paul Jackson volunteered to maintain a data base of machine-checked proofs. He requested other participants to send him pointers to such data bases being maintained by their respective groups. Jackson would inform the QED mailing lists about the site of such a data base and how it could be accessed, and later, the contents of this data base.

Considerable interest was expressed in learning about the continued progress being made by different theorem proving and proof checking systems. It was suggested that new developments and progress should be shared with the community by posting them on the QED mailing list.

A concern was raised that the QED mailing list experiences occasional bursts of mail messages. Traffic volume increases considerably, almost to the extent that it becomes difficult for many of us to keep track of the discussion. The mailing list is inactive during most of the time though. How does one maintain a steady flow of contributions to the mailing list? A moderated list was proposed as an alternative, but little need was felt for this change. It was proposed that participants post news items of interest on the mailing list on a regular basis. Kapur volunteered to post the abstracts of the papers appearing in *J. of Automated Reasoning* soon when a new issue comes out. Roman Matuszewski promised to do the same for the Mizar *J. of Formalized Mathematics.* Developers and users of theorem provers and proof checkers were encouraged to post articles on new developments and progress related to QED activities made by their groups.

There was unanimity that both the workshops had been very helpful in bringing researchers together and highlighting most issues related to the QED project. The first workshop was mostly to familiarize with each other's background, work and interests, and how they perceived QED and their work in relation to QED. This was reflected in discussions as well as the round table organization. The second workshop has been oriented towards a more structured presentation of topics of interest and relevance to QED followed by discussions. There was a view that the next

[1]e-mail: kapur@cs.albany.edu

workshop should be in the form of extended abstracts/papers directly contributing to QED, and a proceedings instead of a report could be produced at the end of the workshop. Most participants felt that would be premature, and that it is good to continue with the format of the second workshop with some minor changes.

Piotr Rudnicki agreed to look into the possibility of organizing a workshop at Banff, Canada, sometime in mid May, 1996. No need was felt to change the way the next workshop should be organized.