

Deliverable 3.2:  
A proposal for a common informal language

Mathematical Knowledge Management Network  
MKMnet, IST-2001-37057

## Summary

The present document is the result of **Task 3.2: *A proposal for a common informal language***, of the **Work Package 3: Varied Presentations of Mathematics and Varied User Needs**.

The purpose of this task is to analyse some common ways of writing (informal) mathematics and to propose a common (informal) language.

The sections have been compiled by various partners in the project, using information from other groups when necessary.

The complete report has been collected and integrated by Fairouz Kamareddine, Heriot-Watt University, Edinburgh.

# 1 De Bruijn's verification of mathematics

At the start of Automath project in 1967, de Bruijn wrote (see A.1 of [10]): *The Automath can become an automaton that turns out mathematical theorems in perfect form, with complete proof..... We can distinguish four stages in working with Automath.*

- (1) *C, the creative mathematician, creates something. He writes it down in a style that is clear to himself (or did that long ago). Possibly he has overlooked a number of important details entirely. Filling those gaps often requires much less creativity than the main job did (but sometimes much more!).*
- (2) *R, the reproductive competent mathematician, formulates theorem and proof in a way that is absolutely flawless for the modern mathematician. He has to be well-acquainted with the area, including some of its background. And he should be skilled in axiomatics too.*
- (3) *P, the mathematical programmer, translates his predecessor's text into a number of statements which are digestible for the Automath. He does not necessarily have to understand the mathematical content, but he should be well-trained in handling mathematical patterns of reasoning as well as in handling the codes of Automath. And he should have enough experience for filling the little gaps that R has still left.*
- (4) *A, the Automath, considers P's statements as hints for the construction of a perfect text. P's hints will be accepted by A only if A recognizes them as steps allowed by A's standards. All what A does is perfect. Whether A's production is interesting or relevant depends completely on C, R and P. If all ends well, then A has produced what R intended. Whether that is also what C meant is usually difficult to trace.*

In Automath however, like in other theorem provers and in logic too, the step (1)  $\Rightarrow$  (2) from a mathematical text (written by C in the Common Mathematical Language CML) to a more logical text (written by R in a mixture of CML and logic) is not precisely described. The step (2)  $\Rightarrow$  (3) from the logical text (written by R) to the programmable text (written by P) is also not precisely described but can be made more precise than that of the step from (1) to (2).

For this reason, members of the network collaborated on and published various papers on the subjects of relating and comparing the texts produced by C, R, P and A. This will have implications for all the chain (1)  $\Rightarrow$  (2)  $\Rightarrow$  (3)  $\Rightarrow$  (4) in the sense that we could be moving far closer than ever to

saying that the informal text in (1) is accurately reflected and completed by the logical text in (2) which is accurately reflected and programmed in (3) which is accurately proof checked in (4).

This deliverable consists of a collection of these papers which are described in the following subsections.

## 1.1 An edited book on informal languages and techniques

The edited book [8] contains a list of articles tackling languages and translations of mathematics which are a useful contribution to how one can classify an informal language of mathematics and what are the needed ingredients to pass from an informal style of mathematics to a formal one. In particular:

- **Transitive closure and inductive reasoning** [1] This paper by Avron argues for the special role of the transitive closure operation for understanding inductive definitions and inductive reasoning. Specifically the idea is to focus on a logic obtained from first order logic by adding an operator for defining the transitive closure of any defined relation. Such results are useful for automated reasoning and for proof planning. In particular, mathematical authors usually leave many proofs incomplete and depend on the reader to complete these proofs. Completing these proofs requires filling in many gaps. This paper can be helpful since inductive reasoning is at the heart of proof planning.
- **Recent Results in Type Theory and their Relationship to Automath** [4] Nuprl has been used to represent a large library of mathematics. Automath, was the first computer system to represent and automate mathematics. Both Nuprl and Automath start from an informal text of mathematics and represent it fully formalised in the corresponding system. For this to work well, it is important to know how different informal parts can be formalised. This paper by Robert Constable gives a concise introduction to two recent results from his Nuprl group at Cornell.
  1. The introduction of a ‘dependent intersection’ type by A. Kopylov. This permits the composition of dependent record types from types for records with single fields, and yields the expected subtyping relationship between related record types.
  2. A generalisation of Nuprl’s notion of quotient type by A. Nogin. This generalisation appears to permit more ready use of quotient types

in constructive proofs where the computational content (proof term) is mostly implicit and synthesised from the proof structure.

- **A Mathematical model for biological memory and Consciousness** [3] This article by N.G. de Bruijn explains a set of ideas on models and metaphors that try to assist in filling the gap between mind and matter. Controversially or not, de Bruijn argues that mathematical thinking is not computational and that it is only the final formal representation of mathematics that may be checked algorithmically. This gave him the insight that thinking is jigsaw puzzling and not algorithmic computation. His work on computerized checking of mathematics where he had to deal with the complete path from intuitive mathematical thinking to its formal computer-checkable representation, played a major role in his position on mind and memory.
- **Data Structures for Trees and Tree Contexts, And a Uniform Sharing Functor, As Design Issues for Symbolic Computation Systems** [7] This article by Huet describes two programming techniques with uses in the manipulation of symbolic tree-structured objects. One, ‘The Zipper’, decomposes a tree at a particular subtree of interest, representing the one-hole context in which it stands as stack both giving the hole-to-root path and cacheing the other subtrees which branch off that path. The other, the ‘Share functor’, provides a uniform hashing structure, instances of which may be used to compress tree structures by ensuring that common subtrees are not duplicated in memory.
- **Proof Development with OMEGA:  $\sqrt{2}$  is not rational** [12] This paper by J. Siekmann, C. Benzmueller, A. Fiedler, A. Meier, M. Pollet gives a good overview and motivation for the OMEGA system at a well-chosen level of detail and helps readers to understand the basic ideas and architecture of the system. Moreover, going through a detailed case study certainly helps to give a realistic impression of the system. OMEGA is a fascinating system which is at the cross-road between formal and informal mathematics. It is at the informal level because it does not fill all the cumbersome formal details that fully formalised texts need to have. OMEGA can however move from the informal to the formal by filling in proofs and concepts. This style of OMDOC make it well-placed as a system for both teaching mathematics and formalising mathematics. A central role is played by the proof plan data structure (PDS) which manages proofs at different levels of abstraction. During proof search and interactive proof development,

the PDS is manipulated by agents until a complete plan is found, which in the end is fully expanded into a natural deduction proof which can be verified by Omega. Furthermore, Omega provides access to external systems: computer algebra systems, automated theorem provers, model generators and constraint solvers. These systems can be charged with subproblems that arise during proof search. The outputs of these systems have to be translated into omega proofs to ensure soundness of the whole proof development. Languages of informal mathematics have a great deal to learn from the OMEGA experience.

- **Termination in ACL2 Using Multiset Relations** [11] This article by J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo and F.-J. Martin-Mateos presents ACL2 solutions to some standard, although challenging, problems. This is all made somewhat difficult because ACL2 is a quantifier-free logic, and dispenses with helpful things like lambda-abstraction. The authors have formalized and proved the well-known theorem that the multiset relation induced by a well-founded relation is also well founded. The ACL2-proof of this theorem allows the authors to effectively construct multiset relations in proofs which are then (known to be) well founded by construction. In this way, the authors proved the correctness of a program transformation technique, the termination of McCarthy's 91-function, and the Diamond (or Newman) Lemma. All these three proofs relied on the use of (well-founded) multiset relations. The main contribution of the paper is that it shows that such mathematical theorems can also be formalized and proved in a system like ACL2 which has a much more restricted logic than theorem provers like Isabelle, HOL, PVS, etc., which use higher-order logic. Typically, ACL2 is rather applied for software or hardware verification, not in mathematical areas. Then, the aim is to perform routine verification steps automatically and to require user interaction only for creative steps. For that reason, ACL2 works just with a quantifier-free first order logic, but its degree of automation is much higher than for systems with a more powerful logic. One would therefore expect that ACL2 would be less suitable for formalizing mathematics. In fact, (at least) two of the case studies presented in the paper have been formalized in systems like HOL, PVS, etc. before. The paper demonstrates that interesting mathematical properties in this area can indeed also be formalized and proved in ACL2.
- **Theory and Models of the pi-Calculus using Fraenkel-Mostowski Generalised** [6] In this article, Gabbay presents a generalisation of FM

theory within higher-order logic, and applies it to model the syntax and operational semantics of the pi-calculus. The generalisation of FM theory is to abstractly characterise the type A of atoms within HOL, and thereby admit notions of "small" and "large" other than finite/cofinite. The models of the pi-calculus are then inductive constructions within FM, using the freshness quantifier to support a very natural reasoning style.

- **Towards an Interactive Mathematical Proof Mode** [2] This article by Henk Barendregt outlines a declarative proof language for type theory, and also hints at an interactive development mode for that language. The aim of the paper is to convince implementors of mathematical assistants to make systems in such a way that formalising proofs becomes natural. This paper is another excellent attempt at reflecting on informal mathematics instead of getting immersed in fully formalised mathematics. The paper argues that we need to attract the user to use the system and to do so, the presentation of mathematics needs to be as interesting as possible.
- **Hoare Logic with Explicit Contexts** [5] This article by Michael Franssen proposes an approach to combining Hoare logic with a typed lambda calculus. The author presents several ideas to design a Hoare Logic conforming to the de Bruijn criterion i.e. having its derivations mechanically checked. The main idea is to explicitly introduce in programs proofs involved in the consequence rule, using a new construction "fake p" where "p" is a proof term (proofs are conducted in a typed lambda-calculus). Several properties of this logic are investigated (forward/backward inference, decidability of checking, etc.) Another idea is to have a clear separation between objects to be used within programs and specifications, or in only one of the two. Triple of contexts, called Hoare contexts, are introduced for this purpose.

## 1.2 Weak Type Theory as an informal language of mathematics

The paper [9] provides a syntax and a syntax-driven derivation system for a formal language of mathematics together with its metatheory and a number of illustrative examples. This language is a refinement of de Bruijn's Mathematical Vernacular (MV). As a result, it satisfies the properties that de Bruijn intended for his MV:

- It is as faithful as possible to the mathematician’s language yet is formal and does not allow ambiguities.
- It is close to the usual way in which mathematicians express themselves in writing.
- It is relatively easy to use and has a high degree of reliability in the sense that it gives a true image of what the mathematician intends to say.

Like de Bruijn’s MV, it satisfies the following attractive conditions:

- It has a syntax based on linguistic categories instead of set/type theoretic constructs as is usually done in the foundations of mathematics.
- It has a derivation system which helps in restricting its usage to cases which obey a number of ‘natural’ linguistic requirements.
- It uses the *book* as a central entity to derive judgements like  $\vdash B : \text{book}$  instead of the usual typing judgements of type theory like  $\Gamma \vdash B : C$ .

Unlike MV however, the new language has a precise abstract syntax (rather than the open ended style of MV). Again, unlike MV, its derivation rules resemble those of modern type theory enabling us to establish important desirable properties such as strong normalisation, decidability of type checking and subject reduction. The derivation system allows one to establish that a book written in this new language is well-formed following the syntax of language, and has great resemblance with ordinary mathematics books.

## 2 Conclusions

The famous mathematician Frege was frustrated by the informalities of the common mathematical language CML: *...I found the inadequacy of language to be an obstacle; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain precision... (Begriffsschrift)*. In 1879, he wrote the *Begriffsschrift*, whose *first purpose is to provide us with the most reliable test of the validity of a chain of inferences* (again, see *Begriffsschrift*, Preface). Then he wrote the *Grundlagen* and *Grundgesetze der Arithmetik* where he argued that mathematics is a branch of logic and described arithmetic in *Begriffsschrift*. Russell wrote a letter to Frege informing him of a paradox in Frege’s work and his own. To avoid the paradox, Russell used *type theory* in

the famous *Principia Mathematica* where mathematics was founded on logic. Advances were also made in set theory, category theory, etc., each being advocated as a better foundation for mathematics. But, none of the logical languages of the 20th century satisfies the criteria expected of a language of mathematics. A logical language does not have mathematico-linguistic categories, is not universal to all users of mathematics, and is not a satisfactory communication medium:

- Logical languages make fixed choices (first versus higher order, predicative versus impredicative, constructive versus classical, types or sets, etc.). But different parts of mathematics need different choices and there is no universal agreement as to which is the best formalism.
- A logician writes in logic their understanding of a mathematical-text as a formal, complete text which is structured considerably unlike the original, and is of little use to the *ordinary* mathematician.
- Mathematicians do not want to use formal logic and have for centuries done mathematics without it.

So, mathematicians kept to CML. In this deliverable, we argued for an alternative to CML which avoids some of the features of the logical languages which made them unattractive to mathematicians. We looked at some of those alternatives that members of the network provided. We believe that these alternatives will open a new useful era of collaboration between mathematicians and logicians. This bridging between mathematics and logic can also reach computer science and proof checking. In 1967 the famous mathematician de Bruijn began work on logical languages for complete books of mathematics that can be checked by machine. People are prone to error, so if a machine can do proof checking, we expect fewer errors. Most mathematicians doubted de Bruijn could achieve success, and computer scientists had no interest at all. However, he persevered and built Automath (AUTOMated MATHeMatics). Today, there is much interest in many approaches to proof checking for verification of computer hardware and software. Many theorem provers have been built to mechanically check mathematics and computer science reasoning (e.g. Isabelle, HOL, Coq, etc.). In practice, a CML-text is structured very differently from a computer-checked text proving the same facts. Making the latter involves extensive knowledge and many choices:

- First, the needed choices include:
  - The choice of the underlying logical system.

- The choice of how concepts are implemented (equational reasoning, equivalences and classes, partial functions, induction, etc.).
  - The choice of the formal system: a type theory (dependent?), a set theory (ZF? FM?), etc.
  - The choice of the proof checker: Automath, Isabelle, Coq, PVS, Mizar, etc.
- Any informal reasoning in a CML-text will cause headaches as it is hard<sup>1</sup> to turn a big step into a (series of) syntactic proof expressions.
  - Then the CML-text is *reformulated* in a fully *complete* syntactic formalism where every detail is spelled out. Very long expressions replace a clear CML-text. The new text is useless to ordinary mathematicians.

Thus, automation is user-unfriendly for the ordinary mathematician/computer scientist. It is the hope that our proposal for an informal language may help in dividing the jump from informal mathematics to a fully formal one into smaller more informed steps.

## References

- [1] A. Avron. Transitive closure and inductive reasoning. In [8]. November 2003.
- [2] H. Barendregt. Towards an Interactive Mathematical Proof Mode. In [8]. November 2003.
- [3] N.G. de Bruijn. A Mathematical model for biological memory and Consciousness. In [8]. November 2003.
- [4] R. Constable. Recent Results in Type Theory and their Relationship to Automath. In [8]. November 2003.
- [5] M. Franssen. Hoare Logic with Explicit Contexts. In [8]. November 2003.
- [6] M.J. Gabbay. Theory and Models of the pi-Calculus using Fraenkel-Mostowski Generalised. In [8]. November 2003.
- [7] G. Huet. Data Structures for Trees and Tree Contexts, And a Uniform Sharing Functor, As Design Issues for Symbolic Computation Systems. In [8]. November 2003.

---

<sup>1</sup> *Tactics* help but give a *track* for the final proof which is not informative nor accessible.

- [8] Fairouz Kamareddine, editor. *Thirty Five years of Automath*, Applied Logic Series, Volume 28. Kluwer Academic Publishers, November 2003.
- [9] F. Kamareddine, and R.P. Nederpelt. A refinement of de Bruijn's formal language of mathematics. To appear in the journal of *Logic, Language and Information*, 2004.
- [10] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*. North-Holland, Amsterdam, 1994.
- [11] J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo and F.-J. Martin-Mateos. Termination in ACL2 Using Multiset Relations. In [8]. November 2003.
- [12] J. Siekmann, C. Benzmueller, A. Fiedler, A. Meier, M. Pollet. Proof Development with OMEGA:  $\sqrt{2}$  is not rational. In [8]. November 2003.