

Deliverable 3.1:
State of the art survey paper on
domain-specific and national variations on
notations

Mathematical Knowledge Management Network
MKMnet, IST-2001-37057

Summary

The present document is the result of **Task 3.1**: *State of the art survey paper on domain-specific and national variations on notations*, of the **Work Package 3**: Varied Presentations of Mathematics and Varied User Needs.

The purpose of this task is to analyse the various notations present, both nationally-specific and subject-specific, in mathematical presentation.

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 Domain-specific and National variations on (formalised) notations

In the world of mathematics, different countries use different notations. Similarly, many modern mathematical notations differ from historical ones.¹ Up to this stage of the project, we concentrated on the formalised use of notations (rather than the informal use) and we therefore produced a (partial) state of the art survey on national variations on notations. Instead of a paper, we produced:

- A number of papers which compare different notations assessing the weaknesses and giving useful extensions [6, 10, 13, 12].
- A collection of papers edited in one special issue (dedicated to the pioneer N.G. de Bruijn) [9] each of which again compares different notations assessing the weaknesses and giving useful extensions.

1.1 Structuring advanced mathematical knowledge in Theorema

The paper of Buchberger et al [4], present results in this area in the Theorema system. In the Theorema system [3], structuring advanced mathematical knowledge bases on hierarchical domains is implemented that is similar to the functor construct in SML but goes significantly beyond this construct in the following respects: the domain carrier can be an arbitrary predicate; the operations in the functor can be arbitrary mathematical definitions, i.e. they are not confined to algorithmic definitions; and the operations of the functors can not only be executed but their correctness can also be proved within Theorema. A major case study on the use of Theorema functors for building up an abstract parametrized version of the author's Groebner bases theory has been delivered in [4].

1.2 How the presentation of mathematics affects its understanding

The paper of [6] analyses theoretically and experimentally the problems students in mathematics have with understanding the subject in relation to the

¹For example Frege's notation has not become as popular as that of Peano. Also, the influential Automath system used a completely different notation for functions and abstractions than modern theorem provers.

way the subject is presented. It is shown how some concepts and notations, for example the concept of "absolute value", have historically developed. It is studied from a historic, didactic and cognitive point of view how different notations, conceptions and formalizations as well as transitions between different systems support or hinder the understanding of the concepts in question. This is confirmed by experiments with several hundreds of students. It is pointed out that changing notation is often necessary and that it is important also to teach the relation between notations.

1.3 How different representations of functions allow different reasoning power about mathematics

The paper [13] describes the role of functions in logic and mathematics and compares the different uses of functions in the historical works of Frege, Russell, and Church. Then, it points out that Church did miss some aspects of function construction and that this explains why theorem provers which are based on Church's lambda notation make many extensions to the lambda calculus in order to represent mathematics more accurately. For example, definitions and parameters are not present in the lambda calculus but are heavily needed and very necessary in the theorem prover Automath. The paper shows that if one extends the lambda calculus with the mathematical notions of definitions and parameters then influential programming languages and proof checkers (e.g., Milner's ML, de Bruijn's Automath and the Edinburgh logical framework LF) can be compared to and placed more accurately in the modern hierarchy of type systems enabling a lot of the modern day machinery and proof techniques to be transported to complex systems like ML and Automath. This work also allows one to explain the distinction between the mathematician (who is interested in *working* with mathematics) and the logician (who is interested in *reasoning* about mathematics).

1.4 The place of the first system for automating mathematics within modern theorem provers

The paper [10] discusses the place of the Automath functions within modern type systems and theorem provers. Automath, takes functions as written in Mathematics (using definitions and parameters) as basic. Modern type theory never discusses definitions and parameters, and is only interested in the abstraction aspects of functions (where there are different abstractors λ and Π). De Bruijn does not even distinguish between λ and Π in his Automath. This paper studies the state of type theory if the notions of de

Bruijn's abstractions were used.

1.5 Comparing different notations of explicit substitutions in theorem provers

The paper of [12] compares the French notation of explicit substitution (à la $\lambda\sigma$) to the British one (à la λs_e) and the proofs of termination of their underlying substitutions, formalising both proofs in the Swedish proof checker ALF. Conclusions are drawn as to which style is better for which application.

1.6 An edited collection on domain-specific and notational variations on notations

The papers of [9] are mostly written by partners in the project, but when necessary, we invited authors who are not partners but who are experts in the field. All the volume however is edited by Professor Kamareddine who is a partner in the project. Furthermore, all the papers in volume [9] are reports on works closely related to this deliverable. In particular, they are works on domain-specific and notational variations on notations. The papers of the edited issue [9] can be described as follows:

- **A Compendium of Continuous Lattices in Mizar** This article by two members of our Polish partners (Bancerek, Rudnicki) presents the impressive effort of the Mizar team to formalize the contents of a well known modern text on lattice theory, "A Compendium of Continuous Lattices" (CCL), by G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott, Springer-Verlag, 1980. The formalization is carried out in the MIZAR system, as part of the even larger "MIZAR Mathematical Library" enterprise. The CCL project started in 1996 and involves a large group of people. The paper is accessible to readers with very little previous knowledge of MIZAR.
- **A proof of GMP Square Root** The article of the members of the French impressive team of Coq (Bertot, Magaud and Zimmermann) deals with a complete proof in the proof assistant Coq of a real algorithm to compute the square root of large integers. Three steps are studied:
 - A formal description of an efficient version of the sqrt program.
 - The correctness of the implementation (written in C) where a description of the conditions under which the algorithm operates

correctly is given. The difficulty resides in the use of imperative languages like C. However, the paper uses a Coq tool called Correctness which produces verification conditions that can be proved from an annotated program.

- The correctness of the memory management where the formalization covers even the details of memory usage and pointer arithmetic. The paper shows that the same scheme can be used for the certification of the program and of the memory management.

- **Automated Proof Construction in Type Theory Using Resolution** This article of our Dutch collaborators (Bezem, Hendriks and de Nivelles) is concerned with translating refutation proofs obtained by resolution to Coq, using reflection and an encoding of the resolution proof in minimal logic. The paper adds automated proof search to type theory, more specifically the paper adds resolution to Coq making type systems usable in real applications. This leads to a system which has the following features:

- A way of representing clauses that admit both a classical resolution strategy and an interpretation in classical dependent type-theory.
- A representation of clauses in minimal logic such that the λ -representation of resolution steps is linear in the size of the premises. A naive translation would be exponential in the length of the resolution proof.
- A translation of resolution proofs into lambda terms, yielding a verification procedure for those proofs.

- **Proof Reflections in Coq** This article of Hendriks describes a formalization of natural deduction proofs for intuitionistic first order logic in the proof assistant Coq. Soundness of the formalization is proved by interpreting logic formulas into formulas in Coq. As an example, the author formalizes Prawitz' permutative conversions and proves the subject reduction property. This paper gives a complete (first-order) formalization of natural deduction for first-order logic using analytic judgements. Features of this work include a Coq implementation and a library which can be reused for several purposes such as the investigation of the meta-theory of deduction systems and the automation of proof-search of first-order theorems.

- **External Rewriting for Skeptical Proof Assistants** This article of Nguyen, Kirchner and Kirchner describes an approach to incorpor-

ate term rewriting in proof assistants based on constructive type theory such as Coq. It contributes to the expanding field of hybrid theorem provers that try to combine strong points of different proof assistants in order to improve their performance and enhance their efficiency. The paper proposes a method to describe rewriting proofs in a calculus called ELAN, then to translate ELAN proofs to Coq proofs. The translation is proved to be sound, thus we can safely use rewriting techniques in ELAN to prove an equation in Coq. The idea is that the task of subgoals rewriting is delegated to ELAN. Then, ELAN proofs are sent back to Coq for verification. In order to carry out this movement between Coq and ELAN, the authors use a calculus $\rho\sigma$ which has explicit substitutions and explicit rewriting.

- **Algorithms and Proofs Inheritance in the Foc Language** This article by our French partners (Prevosto and Doligez) presents the programming language FOC, which is specifically designed to develop computer algebra systems. This language incorporates facilities to prove properties of the developed programs. Object-oriented style inheritance allows mathematical structures to be specified and extended in a natural way. After presenting the core set of features of FOC, the authors describe the static analysis, which rejects inconsistent programs. Static analysis can check properties of specifications which include information such as when proofs of theorems or termination proofs for functions can be inherited. The programs written in FOC are translated to OCAML to be compiled and the system Coq is used to verify the proofs.
- **A New implementation of Automath** This article of our collaborator Wiedijk resurrects Automath and is hence of historical value. The paper describes a new implementation (called ‘aut’) of de Bruijn’s Zandleven Automath checker from the seventies. It describes in some detail the features of aut and was written to restore a damaged version of Jutting’s translation [15] of Landau’s book [14]. Wiedijk establishes that aut is quite fast, even when compared to current theorem prover systems (aut can check the translation of a full book in under a second). The implementation covers different Automath dialects. In addition to the implementation, Wiedijk discusses different aspects of the position of the type systems of Automath within modern type systems. For example, Wiedijk discusses the implications in Automath of having the type of a λ -expression be itself again a λ -expression. In a pure type system the type of a λ -expression is a *product type* or a Π -expression.

- **A comparison of Mizar and Isar** This article of Wenzel and Wiedijk compares two declarative proof systems: Mizar and Isar. The paper highlights the main differences between Mizar and Isar. It begins with a literature review describing other attempts to imitate the Mizar system. Some similarities between Mizar and Isar are described. For the purpose of comparison, a well-known theorem is proved using both systems, namely Euclid's proof that there are infinitely many primes. Eighteen differences are presented which illustrate that:
 - Both systems have their strong points from a user's point of view (so it's not the case that one is generally "better" than the other).
 - Both systems can be improved by learning from the strong points of the other system.

1.7 An edited collection on Mathematical notations

In July 2003, we held a workshop on the interaction of Mathematics, Logic and Computation (at which we also celebrated the 85th anniversary of N.G. de Bruijn, the founder of the first influential system of formalising and proof checking mathematics). We have also gathered an excellent selection of papers on the topic of Mathematical formalisations and notations which we have just edited in a special issue of ENTCS [5]. The papers in the issue are again written by partners in the project and can be described as follows:

- **Automath and Pure Type Systems** [11] This paper by Kamareddine, Laan and Nederpelt, tries to bridge the gap between Automath-like type systems and "modern" type systems like the Calculus of Constructions and the family of Pure Type Systems. This is done by casting the system AUT-68 into lambda-68, a system close to a Pure Type System. Important aspects of Automath systems that are missing in PTSs (though sometimes present in their implementation as a proof assistant) are definitions and parameters and the use of "books" and "lines" to incrementally define and construct formal mathematics. This paper is a good attempt at presenting the Automath ideas in a formal way to an audience used to present day systems. There is a formal translation from AUT-68 to lambda-68, for which a soundness result is proven.
- **Parameterization for theorem proving** [8] In this paper, the author Jojgov presents an extension of pure type systems with abstractions over parameterized variables and shows that this extension satisfies the necessary theoretical properties. The author's other aim is also to

show that such a calculus can be used to model states and transitions between states in a theorem prover. This is a useful progress since theorem provers depend on incomplete proofs and information and the author's work can help fill the gap in the work on holes (or incomplete proofs) in theorem provers.

- **Polymorphic type checking for the ramified theory of types of Principia Mathematica** [7] This paper by Holmes is another relevant paper where a very historical system (that of Russel and Whitehead's Principia's) has been formalised. This paper builds on an earlier work by Kamareddine, Laan and Nederpelt and is another useful exercise which shows that it is not trivial to formalise the intended intuitions of the mathematician and unfortunately now it is too late to ask Russell and Whitehead.
- **A simple canonical representation of rational numbers** [2] This paper by Bertot develops a representation of reduced rational numbers based on traces of Euclids algorithm for reducing them. This is an interesting paper describing an inductive definition of the rational numbers. The rationals are usually described as a quotient of the set of pairs of integers (or an integer and a rational). In the definition of operations and relations on the rationals (which is done on representants of equivalence classes), this always requires a proof that the operation (or relation) is compatible with the equivalence relation. In pure mathematics, this may not be a big problem, as one "abstracts away" from this and leaves the details implicit (often unverified). When formalizing the rationals in a theorem prover, especially when constructing them to be able to define real executable algorithms on them, the quotient construction complicates things. The present paper describes an inductive definition of the rationals which allows relatively simple definitions of operations and relations on them. The work has been formalized in Coq, so an inductive type of rationals is defined together with functions defined by recursion over the inductive type.
- **On the structure of Mizar Types** [1] This paper by Bancerek describes (part of) the structure of the Mizar type system. This paper is a valuable contribution to the representation format of one of the most extensive corpora of formalized mathematics.
- **Other papers** of [5] describe other styles and proofs in mathematics and give us better ways to reach difficult results.

2 Conclusion

The studies carried out and the results obtained in this deliverable illustrate clearly the vast amount of notations in the mathematical knowledge management field and the huge task that is facing us in order to bridge these notations and to allow findings to be exported from one notation to another. Our studies show that indeed a lot of benefits can be obtained from bridging notations. Moreover, despite having been very active in this deliverable, carrying out significant studies and disseminating the results at conferences, in books and in journals, much remains to be done and we depend on the support of funding bodies to be able to tackle this very important area. The support of the European Union has been greatly appreciated. Without that support, we would not have been able to make these useful contributions.

References

- [1] G. Bancerek. On the structure of Mizar Types. In [5]. July 2003.
- [2] Y. Bertot. A simple canonical representation of rational numbers. In [5]. July 2003.
- [3] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The Theorema Project: A Progress Report. In: Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, August 6-7, 2000, St. Andrews, Scotland), M. Kerber and M. Kohlhase (eds.), A.K. Peters, Natick, Massachusetts, pp. 98-113.
- [4] B. Buchberger. Groebner Rings in Theorema: A Case Study in Functors and Categories. Technical Report SFB 2003-26, Special Research Area "Scientific Computing", Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, November 2003.
- [5] H. Geuvers and F. Kamareddine, editors. *Special Issue on Mathematics, Logic and Computation, Electronic Notes in Theoretical Computer Science*, volume 85(7). Elsevier, July 2003.
- [6] A. Gagatis. A Multidimensional Approach to Understanding Mathematics. *Proc. 3rd Mediterranean Conf. on Math. Ed.*, 53-69. Athens 2003.
- [7] R. Holmes. Polymorphic type checking for the ramified theory of types of Principia Mathematica. In [5]. July 2003.

- [8] G. Jojgov. Parameterization for theorem proving. In [5]. July 2003.
- [9] F. Kamareddine, editor. *Special Issue on Mechanizing and Automating Mathematics: In honour of N.G. de Bruijn*, *Journal of Automated Reasoning*, volume 29 (3-4). Kluwer Academic Publishers, December 2002.
- [10] F. Kamareddine. *On Functions and Types: A Tutorial*, volume 2540 of *Lecture Notes in Computer Science*, pages 74–93. Springer Verlag, November 2002.
- [11] F. Kamareddine, T. Laan and R. Nederpelt. Automath and Pure Type Systems, appears in [5]. July 2003.
- [12] F. Kamareddine, and Haiyan Qiao. Formalizing Strong Normalization Proofs of Explicit Substitution Calculi in ALF. *Journal of Automated Reasoning*, 30:59-98, January 2003.
- [13] F. Kamareddine, L. Laan, and R. Nederpelt. Revisiting the notion of function. *Algebraic and Logic Programming*, 54:65–107, January 2003.
- [14] E. Landau. *Grundlagen der Analysis*. Leipzig, 1930.
- [15] L.S. van Benthem Jutting. A Translation of Landau’s “Grundlagen” in AUTOMATH. Technical report, 1976.