On Communicating Proofs in Interactive Mathematical Documents

Olga Caprotti and Martijn Oostdijk

Eindhoven University of Technology
The Netherlands
{olga,martijno}@win.tue.nl

Abstract. There is a wealth of interactive mathematics available on the web. Examples range from animated geometry to computing the n^{th} digit in the expansion of π. However, proofs seem to remain static and at most they provide interaction in the form of links to definitions and other proofs. In this paper, we want to show how interactivity can be included in proofs themselves by making them executable, human-readable, and yet formal. The basic ingredients are formal proof-objects, OpenMath-related languages, and the latest eXtensible Markup Language (XML) technology. We exhibit, by an example taken from a formal development in number theory, the final product of which we believe to be a truly interactive mathematical document.

Keywords: Interactive mathematical documents, Formal mathematical proofs, Type theory, Markup languages.

1 Introduction

One may broadly classify the many examples of online mathematical documents promising interactivity into two categories: textual documents that are hyper-linked and allow readers to consult several pieces of information by comfortably clicking through the links, and documents that are interfaces to software agents performing computations (for instance, to visualize graphically a surface, to solve a system of equations or to find references of papers in which some integer sequence occurred). Both kinds of documents are considered interactive in that the user is actively involved in producing the final reading material. Unfortunately each document has its own notation and if mathematical objects are used, then they can hardly be directly utilized in a different setting. In this form, the mathematical knowledge cannot be shared.

Although JAVA applets and cgi-scripts have provided the support for embedding graphical and computational facilities into an interactive mathematical document, little interactivity is currently available in proofs. One area in which applets have actually been used in "proofs" is planar geometry [25] where they can be applied naturally. If we adhere to the idea that the essence of doing mathematics is proving, then we must conclude that mathematical knowledge is still

J.A. Campbell and E. Roanes-Lozano (Eds.): AISC 2000, LNAI 1930, pp. 53–64, 2001.

poorly communicated interactively. On the other hand, symbolic computation systems, like computer algebra packages and automated proof environments, can play a key role in supporting the development of interactive proofs, to generate the content, validate it, and act as back engines during online presentation.

In this paper we present a possible approach. We consider tactics-based proof assistants, in the tradition of the AUTOMATH project [23], thus excluding resolution-based theorem provers. We focus on type-theoretical proof assistants such as Coq, Lego, and NuPRL [11, 22, 10], yet some of the technologies involved can be applied to proof plans in the general sense.

The main appeal of using proofs developed in type-theoretical systems is that proofs are terms in the formal language of the system. Proving occurs by interaction with the system through specialized user-friendly interfaces [20] designed to produce formal proofs. These formal proofs are often too detailed and become hard to read. This explains the many efforts made in producing more natural descriptions of the resulting proofs [12, 2, 21, 17]. Still, these efforts produce only "flat text" akin to conventional informal proofs and not directly suited for supporting interaction. In fact, such text is meant to be read by humans and not by programs. For interactive use, a published proof also needs to access the original formal proof that produced the text; the flat text version of the proof alone is not enough.

In our approach, presentation and content are kept distinct. The formal proof is used as content, and multiple views, i.e. multiple presentations of the same content, are generated. In order to do so, the formal proof is encoded using OpenMath [9], a standard markup language for mathematical content. The overall mathematical document is generated automatically in the OpenMath Document [19] format. This allows the inclusion of OpenMath objects such as formal proofs, flat text containing informal mathematics and tactic scripts. In this way, sharing of mathematical knowledge and transparent interaction with computational tools are achieved. In the paper we present some of the technologies that we have developed and that are currently being used for the next version of "Algebra Interactive!" [8, 6].

The paper is structured as follows. Section 2 discusses the different options for including proofs in interactive mathematical documents. In Section 3, some of the basic knowledge needed to understand type-theoretical theorem proving is briefly recalled using an example. The technologies involved in bringing proofs online are described in Section 4. The concluding remarks are found in Section 5.

2 Proofs in Interactive Mathematical Documents

In this section we study the options for embedding proofs within an interactive mathematical document.

The first and most obvious is to include a textual version of the proof in natural language, similar to proofs in traditional (paper) mathematical texts. Indeed, one of the first tools used to publish mathematics on the web was a converter from TEX to HTML. Nowadays, most word processing software is able

	NL Text	Tactics Scripts	Proof Objects
OM encodable	-	-	+
mathematical object	-	-	+
readable	+	+/-	-
machine checkable	-	+/-	+

Table 1. Pros and cons of different proof embedding options.

to output HTML and in some cases the mathematics is not replaced by an image (a gif file) but by its presentation in the MathML language [3].

Although this form of presentation does contain structure, for example references to lemmata or definitions, the format is usually plain text. Because it consists of mathematical vernacular, it is very easy to read but hard to reconstruct a fully formal proof from it. Having the mathematical content semantically marked-up is desirable for two reasons. First, it can be communicated to computational software such as a computer algebra system or a proof checker. Second, the structure can be used to open different views on the proof. Allowing the user to change views is one form of interactivity we achieve in our approach.

The second way to embed proofs is to include high-level tactics scripts which serve as input for theorem provers. This approach is opposite to the former one in that it takes the notion of proof from a theorem prover, i.e. a formal system, instead of from informal mathematics. Such scripts are system dependent and, in general, it is easy to communicate the proof to the specific system and have it checked. Users of the system usually regard tactics scripts as real proofs and do not find it difficult to understand them even though they show only one half of a dialogue. In recent work, high-level tactic scripts are used to produce natural language verbalization of the proof [17].

The third way to embed proofs is to include a formal proof, for instance a derivation tree. In a type-theoretical setting, the corresponding lambda term is a good candidate for the formal representation of the proof. Because it is a mathematical term, it can be encoded using a markup language for mathematics and it or parts of it can be communicated to computational software in a standard way. A major drawback to this option is that formal proofs contain many details and hence are not very readable, moreover their marked-up encoding becomes even larger. However, this drawback can be overcome by creating suitable views on the formal tree leaving out the undesired details and adding information to clarify the formal structure, see for instance our natural language view in Figure 1.

Table 1 sums up the pros and cons of the above three options.

In this paper we propose to mix the different options, thus retaining the readability of textual proofs, the ability to step through the proof interactively, and the rigor of formal proofs. Type theoretical systems are based on the "propositions as types" paradigm which is recalled in the next section through examples.

3 Example of a Formalization in Type Theory

In this section we briefly sketch the process of formally developing a mathematical theory in a proof assistant. The examples we use arise during the formalization, done in Coq, of Pocklington's Criterion [24, 7, 4] for finding whether a positive number is prime.

In the type theory of Coq, mathematical concepts are encoded as typed expressions. For instance, one can introduce the notions of division and primality of natural numbers by the following definitions. `Divides` is defined in the usual way as a predicate on two natural numbers n and m (usually denoted as $n|m$), and `Prime` is defined as a predicate on a natural number n.

```
Definition Divides: nat -> nat -> Prop :=
  [n,m:nat](EX q:nat | m=(mult n q)).

Definition Prime: nat -> Prop :=
  [n:nat](gt n (1)) /\ (q:nat)(Divides q n) -> q=(1)  q=n.
```

Besides concepts from the mathematical object language, also concepts from the meta language such as theorems and proofs are encoded as typed expressions. This principle is called the *Curry-Howard correspondence*, also known as *propositions as types*.

A proof is encoded as an object which has, as its type, the encoding of the statement of the theorem. A typed expression which represents a proof is called a *proof-object*. If an expression of type `Prop` has more than one inhabitant, those inhabitants represent different proof-objects of the proposition. If it has no inhabitants, then it cannot be proved.

Consider as example the observation that if all prime divisors of a positive number n are greater than $\sqrt{n}$, then the number n is prime. This can be stated in Coq as a new theorem (`primepropdiv`) that uses library functions (`mult` and `gt`) and newly defined predicates (`Divides` and `Prime`):

```
Lemma primepropdiv:   (n:nat)(gt n  (1))          ->
                      ((q:nat)(Prime q)           ->
                              (Divides q n)       ->
                              (gt (mult q q) n)) ->
                      (Prime n).
```

To assist the user in constructing proof-objects, Coq uses a high-level language of *tactics*. A user of Coq, trying to prove `primepropdiv`, would be presented with the initial goal and interactively arrive at the following script.

```
Intros. Elim (primedec n). Intro. Assumption.
Intros. Elim (nonprime_primewitness n).
Intros. Elim H2. Intros. Elim H4. Intros.
Elim H6. Intros. Elim (le_not_lt (mult x x) n).
Assumption. Unfold gt in H0. Apply H0.
Assumption. Assumption. Assumption. Assumption.
```

The script uses lemmata from the Coq `arith` library and previously proven lemmata, like `primedec` which proves the decidability of the `prime` predicate and `nonprime_primewitness` which proves that non-prime numbers greater than 1 have prime-divisors. The proof-object `primepropdiv` is given below.

```
[n:nat;
H:(gt n (1));
H0:((q:nat)(Prime q)->(Divides q n)->(gt (mult q q) n))]
(or_ind (Prime n) ~(Prime n) (Prime n) [H1:(Prime n)]H1
[H1:(~(Prime n))]
(ex_ind nat
[d:nat](lt (1) d) /\ (le (mult d d) n) /\ (Divides d n) /\ (Prime d)
(Prime n)
[x:nat;
H2:((lt (1) x) /\ (le (mult x x) n) /\ (Divides x n) /\ (Prime x))]
(and_ind (lt (1) x) (le (mult x x) n) /\ (Divides x n) /\ (Prime x)
(Prime n)
[_:(lt (1) x);
H4:((le (mult x x) n) /\ (Divides x n) /\ (Prime x))]
(and_ind (le (mult x x) n) (Divides x n) /\ (Prime x)
(Prime n)
[H5:(le (mult x x) n); H6:((Divides x n) /\ (Prime x))]
(and_ind (Divides x n) (Prime x) (Prime n)
[H7:(Divides x n); H8:(Prime x)]
(False_ind (Prime n)
(le_not_lt (mult x x) n H5 (H0 x H8 H7))) H6) H4) H2)
(nonprime_primewitness n H H1)) (primedec n))
```

If available, such a proof-object can be used in an interactive document describing Pocklington's Criterion in several ways. First of all, it provides evidence of the truth of the assertion it proves: such a term can be easily checked by Coq to be of type `primepropdiv`. If it is encoded in a system-independent standard language such as OpenMath, then it can be shared. Moreover, it can also be used to produce a natural language view of the proof it represents since in its current form it is not readable. Part of our technology includes a tool that, when given a context and a proof object, produces a natural language view of the associated proof. The tool can interactively adapt the level at which the proof is displayed by collapsing or expanding certain sentences. The major technologies involved are described in Section 4.

4 Enabling Technologies

This section introduces the core technologies developed to support our approach to proofs in interactive mathematical documents. The natural language view is strongly connected to formal proof objects based on type theory. OpenMath and related technologies enable the representation of structured mathematical information such as a proof term, its context, a tactics script and natural language explanations.

4.1 The Natural Language View

The natural language viewer, described in [**?**], extends the standard algorithm for translating Coq proof-objects presented in [13]. The input is a Coq context

and a proof-object. The output obtained by the standard algorithm is a detailed natural language proof. Our implementation takes into account requirements for interactivity and improves upon it in two ways.

First, instead of producing flat text, the final presentation is an adjustable view. It is generated from an object which is tightly connected to the original formal proof-object. This intermediate object contains both natural language parts and formal parts. When rendering such an object on the screen, the formal parts of a sentence may be treated differently. For example, whenever the name of a definition occurs, the renderer produces a hyperlink to the place in the context where the definition is introduced.

Second, recursive calls of the translation algorithm result in a sentence which can be expanded or collapsed by the reader, similar to the display of a directory structure in file-browser in modern GUI systems. When the sentence is collapsed, instead of recursively translating the corresponding subproof, the renderer displays the type of the subproof. Figure 1 shows a combination of the natural language view with Fitch-style natural deduction notation.

The translation algorithm is rule based, and is driven by the structure of the proof-object. Some of the rules are given below. Note that the translations uses type inference to guide the translation process. Here $\boxed{M}_\tau$ on the left hand side means "M is of type τ" and $\blacksquare$ (M) on the right hand side means "recursively translate M when the folder is open, display the type of M when it is closed".

$$\boxed{h}_\tau \mapsto \begin{cases} \square \text{ “By” } h \text{ “we have” } \tau \end{cases}$$

$$\boxed{\boxed{M}_{(\forall x:A.B)}\boxed{N}_A}_\tau \mapsto \begin{cases} \blacksquare\ (M) \\ \square \text{ “By taking” } N \text{ “for” } x \text{ “we get” } \tau \end{cases}$$

$$\boxed{\boxed{M}_{(A\to B)}\boxed{N}_A}_\tau \mapsto \begin{cases} \blacksquare\ (N) \\ \blacksquare\ (M) \\ \square \text{ “We deduce” } \tau \end{cases}$$

$$\boxed{\lambda h : \boxed{A}_{\texttt{Prop}}.\boxed{M}_B}_\tau \mapsto \begin{cases} \square \text{ Assume } A \ (h) \\ \blacksquare\ (M) \\ \square \text{ “We have proved” } \tau \end{cases}$$

$$\boxed{\lambda x : \boxed{A}_{\texttt{Set}}.\boxed{M}_B}_\tau \mapsto \begin{cases} \square \text{ “Consider an arbitrary” } x \text{ “}\in\text{” } A \\ \blacksquare\ (M) \\ \square \text{ “We have proved” } \tau \text{ “since” } x \\ \quad \text{“is arbitrary”} \end{cases}$$

The sentences are kept as abstract as possible by not unfolding definitions unless needed, using expected types instead of derived types as done in [12] and by repeated introduction of variables in one sentence.

Many of the usual symbols from mathematics and logic can be defined in type theory. The fact that notions such as the natural numbers and their induction principle are not primitive but definable, is regarded as a strength of type-theoretical systems. In fact, even connectives like $\neg$, $\wedge$, $\vee$, and $\exists$. and relations like $=$ (general Leibniz equality) and $<$ (on $\mathbb{N}$) are all definable in Coq. They are defined in the standard library together with their introduction and elimination rules. In theory, it is sufficient to add to the above rules translation rules for inductive types and their inhabitants. However, the translation algorithm treats some defined objects as primitives in order to choose a translation as close as possible to the language of informal mathematics.

The example from Section 3 uses many lemmata from the standard Coq library `Arith`. Some of these should also be treated as primitive by the natural language viewer since they would be considered trivial in informal mathematics.

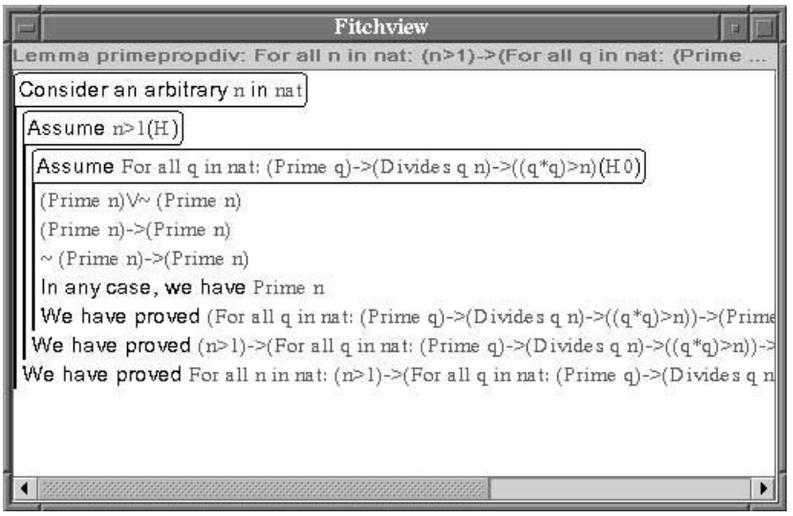

Fig. 1. A Natural Language View of `primepropdiv`

4.2 OpenMath, OpenMath Documents, and MathML

The eXtensible Markup Language (XML) is becoming an increasingly popular choice as source language in which to represent semantically rich information that can be searched, stored and presented in different formats. The World Wide Web Consortium is currently recommending several technologies related to XML and the next generation of browsers will be XML-enabled, namely will be able to directly display XML documents. This section discusses how some of these technologies can be used to provide interactivity to "online" proofs.

OpenMath The predominant XML languages for mathematics are MathML and OpenMath. Ideally these two languages complement each other: MathML-Presentation can be used for presenting mathematical content written in OpenMath. A detailed description of OpenMath is given in [9]. In this paper we assume a certain level of familiarity with the general OpenMath ideas and describe it by examples.

OpenMath is a language for the representation of mathematical content. The symbols used in the OpenMath objects are defined in XML documents called Content Dictionaries (CDs). Official CDs are available for public use from the OpenMath Society [28] but users may also write private CDs of symbols used in own applications. As example, consider the definition of the OpenMath symbol `<OMS cd="pock" name="Divides"/>`, denoted in short by `pock:Divides`, for representing the predicate `Divides` on natural numbers used in the example in Section 3. The formal definition of the example is represented as an OpenMath "defining mathematical property", `DefMP`:

```
<DefMP name="Divides">
<OMOBJ><OMBIND><OMS cd="lc" name="Lambda"/>
       <OMBVAR><!-- n:N, m:N -->
       <OMATTR><OMATP><OMS cd="icc" name="type"/>
                      <OMS cd="setname" name="N"/>
       </OMATP>       <OMV name="n"/>    </OMATTR>
       <OMATTR><OMATP><OMS cd="icc" name="type"/>
                      <OMS cd="setname" name="N"/>
       </OMATP>       <OMV name="m"/>    </OMATTR>
       </OMBVAR>
       <OMBIND><OMS cd="quant1" name="exists"/>
       <OMBVAR><!-- q:N -->
       <OMATTR><OMATP><OMS cd="icc" name="type"/>
                      <OMS cd="setname" name="N"/>
       </OMATP>       <OMV name="q"/>    </OMATTR>
       </OMBVAR><OMA><OMS cd="relation1" name="eq"/>
                     <OMV name="m"/>
                     <OMA><OMS cd="arith1" name="times"/>
                          <OMV name="n"/>
                          <OMV name="q"/>
                     </OMA></OMA>
        </OMBIND></OMBIND>
</OMOBJ>
</DefMP>
```

The formal signature can be given similarly in terms of an OpenMath object.

As mentioned before, a proof-object is a term and as such it can be represented in OpenMath provided some primitive symbols are available in some CD. Symbols for constructing and eliminating inductive types in the Inductive Calculus of Constructions used by Coq are given in the CD called `icc`. Additionally, we have a private CD for symbols that Coq uses in proof-objects and refer to introduction and elimination rules of inference. For example, the Coq symbol `and_ind` in the proof-object in Section 3 is represented by the OpenMath symbol `coq:and_ind` in the private CD `coq` and can be exactly defined in terms of the primitive inductive constructors. There are two reasons for representing the proof-object at a higher level than necessary. Firstly, the extra information conveyed by the specific inference rule used in the proof can be directly used for tuning the natural language presentation of the term. The second reason is

compactness and readability. The proof-object can be easily transformed into one which uses only the primitive inductive constructors.

OpenMath is not designed to express hierarchies of mathematical knowledge since it lacks the mechanisms to relate definitions, theories and theorems. Moreover, the command language used during an interactive session with a computational tool is hard to express formally in OpenMath. Variable declarations and tactic scripts are examples of this limitation and help motivate the introduction of OpenMath Documents.

OpenMath Documents The OpenMath Document Specification (OMDOC) [19], currently under development by Kohlhase and ourselves, is an XML document type definition that can be used to represent general mathematical knowledge the way it is written in lecture notes and in scientific articles, but also in mathematical software like algebraic specification modules or library files of a proof checker. It is being used as source format for the next release of the *Algebra Interactive!* book [8], an interactive textbook used in teaching first year university algebra. OpenMath Documents are intended to be the input format for a knowledge base of mathematics, Mbase [18].

The mathematical objects within an OpenMath Document are expressed using an extension of the XML encoding[1] of OpenMath. Most important, the usage of OpenMath, conveying the semantical content and not the presentational content of the mathematics, offers two major advantages:

- It allows the mathematical knowledge base to use techniques such as pattern matching or unification to implement search, e.g. modulo and equational theory.
- It equips OpenMath Documents with a standard language for the communication among mathematical services, thus making them suitable to be exchanged between systems for symbolic computation and reasoning.

In this paper we focus on using OMDOC for representing and publishing proofs in interactive mathematical documents.

The format of an OpenMath Document provides an interface for proof presentation by allowing fine-grained interleaving between the formally specified part of the proof and the informal, vernacular text. There are several options for writing a proof to an assertion. All of the views discussed in Section 2 may coexist in the same document and, depending on the viewer, presented upon request.

As the techniques for producing natural language descriptions from proof objects or from tactics/proof plans improve, we may well envision that simple conventional "informal" proofs will become reproducible by automatic machinery. For now, it is possible to simply include informal proofs mixing formal OpenMath statements with flat text.

Interactive proofs described by a proof or a tactic script are put directly in an OpenMath Document and become executable once the document is loaded

[1] It contains extra attributes for linking OpenMath subtrees

in a browser. Most of the interactivity available to the reader clicking through a proof is supported by a combination of Javascript and JAVA that is invoked on the correct data by relying on the Document Object Model [14]. Each step in the proof is a request of a performative that is sent to the appropriate software package server or to a broker in a network like [15, 27]. More interesting is the possibility of choosing different mathematical servers depending on the nature of the requested computation. For an example, see [16]. It is well known that, for instance, type-theoretical proof checkers are not very good at equational reasoning [1]. Trying to mimic equational reasoning in such systems produces long and unintelligible proof-objects, not to mention the fact that it requires deep knowledge of the proof assistant for understanding them. This also means that in such cases, an interactive proof gives little insight to the reader and misses the point of the proof.

The OpenMath encoding of the proof-object is also stored in the document directly and can trigger the natural language viewer. Moreover, because it is encoded in standard OpenMath, the proof-object can be exchanged easily among proof checkers implementing similar type theories.

MathML As we said, OpenMath is about conveying content and not about presentation. The same can be said of OpenMath documents. Both are not meant to be read in their XML encoding but transformed to more convenient presentation formats for online browsing.

Two technologies under development that produce customized output formats from an XML input source are based on Cascading StyleSheet (CSS) language and the eXtensible Stylesheet Language (XSL) [26]. Using these, it is possible to convert OpenMath Documents to various flavors of dynamic HTML using MathML-Presentation for the OpenMath objects [5]. Similarly, it is possible to generate LaTeX documents for producing printed version of the material.

5 Conclusions and Future Work

We have presented some of the latest technologies we are developing in order to support true interaction in mathematical documents and in particular in mathematical proofs. The key issue is being able to distinguish content from presentation. The content of a mathematical proof is a formal proof-object encoded in OpenMath, whereas the various presentations are given in an OpenMath document as conventional informal descriptions, or are automatically generated from the proof-object. Computational content of a proof is conveyed by including tactic scripts which become executable by the presentation in browser.

Future work lies in further developing the natural language viewer by including more libraries for primitives, and adding more authoring capabilities allowing user-customized translations. More work needs to be done on the OpenMath document and associated translation tools. In the end, we want to be able to create large interactive mathematical documents semi-automatically by using,

for instance, the knowledge contained in a formal development done in a proof assistant.

Acknowledgements The authors would like to thank Arjeh M. Cohen, Herman Geuvers, Michael Kohlhase, and Manfred N. Riem for suggestions, discussions and inspiration.

References

[1] P. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and Integration of Theorem Provers and Computer Algebra Systems. In J. Calmet and J. Plaza, editors, *Artificial Intelligence and Symbolic Computation: International Conference AISC'98*, volume 1476 of *Lecture Notes in Artificial Intelligence*, Plattsburgh, New York, USA, September 1998.

[2] Yves Bertot and Laurent Théry. A Generic Approach to Building User Interfaces for Theorem Provers. *Journal of Symbolic Computation*, 25:161–194, 1998.

[3] Stephen Buswell, Stan Devitt, Angel Diaz, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt, Stéphane Dalmas, David Carlisle, Roger Hunter, and Ron Ausbrooks. Mathematical Markup Language (MathML) Version 2.0. W3C Working Draft 11 February 2000, February 2000. Available at `http://www.w3.org/TR/2000/WD-MathML2-20000211/`.

[4] O. Caprotti and M. Oostdijk. How to formally and efficiently prove prime(2999). In *Proceedings of Calculemus 2000, St. Andrews*, 2000.

[5] O. Caprotti and M. N. Riem. Server-side Presentation of OpenMath Documents using MathML. Submitted.

[6] Olga Caprotti and Arjeh Cohen. Connecting proof checkers and computer algebra using OpenMath. In *The 12th International Conference on Theorem Proving in Higher Order Logics*, Nice, France, September 1999.

[7] Olga Caprotti and Arjeh Cohen. Integrating Computational and Deduction Systems Using OpenMath. In *Proceedings of Calculemus 99*, Trento, July 1999.

[8] A. M. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive, interactive course material*. Number ISBN 3-540-65368-6. SV, 1999.

[9] The OpenMath Consortium. The OpenMath Standard. OpenMath Deliverable 1.3.3a, OpenMath Esprit Consortium, `http://www.nag.co.uk/projects/OpenMath.html`, August 1999. O. Caprotti, D. P. Carlisle and A. M. Cohen Eds.

[10] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986. URL `http://simon.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html`.

[11] Projet Coq. *The Coq Proof Assistant: The standard library*, version 6.3-1 edition. Available at `http://www.ens-lyon.fr/LIP/groupes/coq`.

[12] Y. Coscoy. A natural language explanation for formal proofs. In C. Retoré, editor, *Proceedings of Int. Conf. on Logical Aspects of Computational Liguistics (LACL), Nancy*, volume 1328. Springer-Verlag LNCS/LNAI, September 1996.

[13] Y. Coscoy, G. Kahn, and L. Théry. Extracting text from proof. In M. Dezani and G. Plotkin, editors, *Proceedings of Int. Conf. on Typed Lambda-Calculus and Applications (TLCA), Edinburgh*, volume 902. Springer-Verlag LNCS, April 1995.

[14] Document Object Model (DOM) Level 2 Specification version 1.0. W3C Candidate Recommendation 07 March, 2000, March 2000. http://www.w3.org/TR/2000/CR-DOM-Level-2-20000307/.

[15] A. Franke, S. Hess, Ch. Jung, M. Kohlhase, and V. Sorge. Agent-Oriented Integration of Distributed Mathematical Services. *Journal of Universal Computer Science*, 5(3):156–187, March 1999. Special issue on Integration of Deduction System.

[16] Online Pocklington Test generating OMDOC. http://crystal.win.tue.nl/~olga/openmath/pocklington/omdoc/.

[17] Amanda M. Holland-Minkley, Regina Barzilay, and Robert Constable. Verbalization of high-level formal proofs. In *Sixteenth National Conference on Artificial Intelligence*, 1999.

[18] M. Kohlhase. MBASE: Representing mathematical knowledge in a relational data base. In *Calculemus '99 Workshop*, Trento, Italy, July 1999.

[19] M. Kohlhase. OMDOC: Towards an Openmath Representation of Mathematical Documents. Technical report, DFKI, Saarbrücken, 1999.

[20] Pascal Lequang, Yves Bertot, Laurence Rideau, and Loïc Pottier. Pcoq: A graphical user-interface for Coq. http://www-sop.inria.fr/lemme/pcoq/, February 2000.

[21] Z. Luo and P. Callaghan. Mathematical Vernacular and Conceptual Well-formedness in Mathematical Language . In *Proceedings of the 2nd International Conference on Logical Aspects of Computational Linguistics 97*, volume 1582 of *Lecture Notes in Computer Science*, Nancy, 1997.

[22] Z. Luo and R. Pollack. *LEGO Proof Development System: User's Manual*. Department of Compter Science, University of Edinburgh, 1992.

[23] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in logic and the Foundations of Mathematics*. Elsevier, 1994.

[24] Paulo Ribenboim. *The New Book of Prime Number Records*. Number ISBN 0-387-94457-5. SV, 1996.

[25] Jurgen Richter-Gebert and Ulrich H. Kortenkamp. *The Interactive Geometry Software Cinderella (Interactive Geometry on Computers)*. Number ISBN 3540147195. SV, July 1999.

[26] W3C. Extensible Stylesheet Language (XSL) Specification. W3C Working Draft, 21 Apr 1999. http://www.w3.org/TR/WD-xsl/.

[27] Paul Wang. Design and Protocol for Internet Accessible Mathematical Computation. Technical Report ICM-199901-001, ICM/Kent State University, January 1999.

[28] OpenMath Society Webs