Integrating Computational and Deduction Systems Using *OpenMath*

O. Caprotti and A. M. Cohen

RIACA, Research Institute for Applications of Computer Algebra
Eindhoven University of Technology
5600MB Eindhoven, The Netherlands

Abstract

The standard *OpenMath* is a crucial ingredient for creating an integrated environment combining systems for computer algebra with proof checkers. *OpenMath* consists of a formal grammar of *OpenMath* objects, their encodings, Content Dictionaries, Phrasebooks and other tools. The *OpenMath* standard allows integration of computational systems of different kind. Here we demonstrate how *OpenMath* works by setting up an environment in which Maple expressions are type-checked by the proof checkers Lego and Coq.

1 Introduction

The observation that computation and deduction are closely related led to several approaches to integrate computer algebra systems with theorem provers. Roughly speaking, there are three directions in which to proceed: include computer algebra capabilities in theorem provers [5,21], include theorem proving capabilities in computer algebra systems [11,7,22,16] or, include computing and theorem proving capabilities in a new framework [10].

Connections between computer algebra software and proof checker/automated deduction systems are usually being developed by partners from one of the two research communities. In this light, it is no surprise that most approaches have the flavor of incorporating one into the other.

For instance, the computer algebra system Mathematica with its powerful rewriting mechanism is taken up in the Theorema project [7] for formal proof checking purposes. And vice versa, proof checkers are inclined to incorporate ι reduction (rewriting of types on the basis of computation whose algorithms are verified) and to let the computations be taken over by a computer algebra system within the formal proof checker.

The use of *OpenMath* enables a much more flexible and, indeed, open approach: both the computer algebra system and the proof checkers are usable

via an independent standard language. Thus, it is up to the user, which tools to use for setting up an integrated system, without having to take a particular one as a fixed starting point. In this way, the idea of verification of results and indeed proof checking gains more solid ground, as one might well replace one system with another having the same functionality.

OpenMath is a language for *representing* and *communicating mathematics* [3]. Originally, it was conceived as a language for all computer algebra systems [15,13]. However, in its latest version, it is equipped for conveying mathematical expressions from all areas of mathematics, for instance logic. Now *OpenMath* can be used to express formal mathematical objects, so that formal theorems and proofs, understandable to proof checkers, can be communicated, as well as the usual mathematical expressions handled by CA systems.

Additionally to the language, *OpenMath* will also provide a range of applications and plug-ins which use *OpenMath* in several areas, in particular electronic publications, mathematical software packages, and the worldwide web. One such tool is presented in this paper and shows how a piece of mathematics, expressed using Maple syntax, can be read and transformed to an *OpenMath* object which can then be shipped off to any software package interfaced to *OpenMath*, for instance GAP, or Maple for computation and Coq or Lego for checking well-typedeness. The environment described in the example assembles several components: a conventional browser to provide the user interface, a custom-built Java applet to take care of control issues, and Coq or Lego to carry out the type-checking.

The paper introduces *OpenMath* in Section 2. In Section 3 a subclass of *OpenMath* objects, called *Strong OpenMath*, is defined for which meaningfulness can be well defined. Type checking of *Strong OpenMath* objects, using the proof checker Lego as a server, is described in Section 4. In the last section, we draw some conclusions and outline future work.

2 *OpenMath*

OpenMath consists of several aspects. Those presented in this section are: the architecture of how *OpenMath* views integration of computational systems, the *OpenMath* Standard, and the *OpenMath* Phrasebooks and tools. The *OpenMath* Standard is concerned with the objects, their encodings, and the Content Dictionaries. The reader is referred to the draft of the *OpenMath* standard [13] for additional missing details.

2.1 OpenMath Architecture

The architecture of *OpenMath* is made up of three layers of representation of a mathematical object: the private layer for the internal representation, the abstract layer for the representation as an *OpenMath* object, and the commu-

nication layer for translating the *OpenMath* object to a stream of bytes. An application dependent program manipulates the mathematical objects using its internal representation, it can convert them to *OpenMath* objects and communicate them by using the byte stream representation of *OpenMath* objects.

It is not in the scope of *OpenMath* to define how communication takes place, or which services are requested in an integrated mathematical environment. It was decided at the beginning stage of development, that *OpenMath* would concentrate on the objects dispatched, namely on the terms of some mathematical theory. Therefore, *OpenMath* is one ingredient among many others that are needed for achieving integration of computational tools.

OpenMath objects are representations of mathematical entities that can be communicated among various software applications in a meaningful way, that is, preserving their "semantics". *OpenMath* provides basic objects like integers, symbols, floating-point numbers, character strings, bytearrays, and variables, and compound objects: applications, bindings, errors, and attributions. Content Dictionaries (CDs) specify the meaning of symbols informally using natural language and, optionally, they might formally assign type information in the signature of the symbols. CDs are public and are used to represent the actual common knowledge among *OpenMath*-compliant applications. A central idea to the *OpenMath* philosophy is that CDs fix the "meaning" of objects independently of the application. In this paper's example, the public CD used to represent the Calculus of Constructions is understood by the Java applet. Because of this, the formal signatures expressed using this CD correspond to lambda terms and as a consequence, the applet described in Section 4 is able to assign precisely the semantical content to a class of *OpenMath* objects that are manipulated.

The integration of *OpenMath* in an application is achieved by a *Phrasebook*, namely an interface program that converts an *OpenMath* object to/from the internal representation. The translation is governed by the CDs and the specifics of the application. The example given in this paper specifies how this translation is achieved in a phrasebook for the system Lego. The same example uses also a phrasebook for Maple to enable the input of mathematical expressions in Maple syntax and to convert them to *OpenMath* objects. These objects are then translated to the corresponding objects in Lego syntax by the Lego phrasebook.

2.2 OpenMath Objects

We now focus on the abstract layer, where mathematical objects are represented by labelled trees, called *OpenMath* objects or *OpenMath* expressions. The formal definition of an abstract *OpenMath* object is given below.

Definition 2.1 *OpenMath objects are built recursively as follows.*

(i) *Basic OpenMath objects: integers, symbols (defined in CD), variables, floating point numbers, character strings, and bytearrays are OpenMath*

objects.

(ii) *If* $A_1, \ldots, A_n$, A, B *and* C *are OpenMath objects,* $v_1, \ldots, v_n$ *are OpenMath variables,* $S_1, \ldots, S_n$ *are OpenMath symbols, then*

$$\textbf{application}(A_1, \ldots, A_n) \qquad (n > 0)$$

$$\textbf{binding}(B, v_1, \ldots, v_n, C) \qquad (n \geq 0)$$

$$\textbf{attribution}(A, S_1\ A_1,\ \ldots\ , S_n\ A_n) \qquad (n > 0)$$

$$\textbf{error}(S, A_1, \ldots, A_n) \qquad (n \geq 0)$$

are OpenMath objects.

All symbols appearing in an *OpenMath* object are defined in a *Content Dictionary* as described in Section 2.4. Application in *OpenMath* is either to be intended as functional application, like in the object **application**(`sin`, x), or as a constructor, like in **application**(`Rational`, 1, 2). Binding objects, like **binding**(`lambda`, x, **application**(`plus`, x, 2)), have been introduced in the latest version of the *OpenMath* standard [13], thus turning *OpenMath* in a language for logic. Attribution can act as either "annotation", in the sense of adornment, or as "modifier". By default, the attributed object cannot be viewed as semantically equivalent to the stripped object. In general, an attributed object's syntactic class is object[1]. However, when the object is a variable, attribution does not change the syntactic class so that attributed variables are considered variables. Attribution is used in Section 3 to express the judgement that says that object A has type t by **attribution**(A, `type` t). Errors occur during the manipulation of an *OpenMath* object and are thus of real interest when communication is taking place. Examples of errors are **error**(`division by zero`), **error**(`damaged encoding`).

2.3 OpenMath Encodings

OpenMath encodings map *OpenMath* objects to byte streams that can be easily exchanged between processes or stored and retrieved from files.

Two major encodings supported and described by the *OpenMath* standard are XML and binary. The first encoding uses only ISO 646:1983 characters [2] (ASCII characters) and is "XML compatible", thus it is suitable for sending *OpenMath* objects via e-mail, news, cut-and-paste, etc and for being further processed by a variety of XML tools. For instance the encoding of **binding**(`lambda`, x, **application**(`sin`, x)) is:

```
<OMOBJ><OMBIND><OMS cd="fns" name="lambda"/>
               <OMBVAR><OMV name="x"/></OMBVAR>
               <OMA><OMS cd="transc" name="sin"/>
                    <OMV name="x"/>
```

[1] Notice in particular that the syntactic class of an attributed symbol is object. This technicality avoids nasty nesting in attribution objects.

```
                    </OMA>
</OMBIND></OMOBJ>
```

From this encoding, it can be read off that the symbols (tagged by `OMS`) `lambda` and `sin` are defined CDs called `fns` and `transc` respectively. The elements `OMA`, `OMBIND`, and `OMV` identify, respectively, application, binding and variables.

The second encoding is an ad-hoc binary encoding meant to be used when compactness is crucial, for instance in interprocess communications over a network. A detailed description of the encodings, and in particular the Document Type Definition (DTD) for the XML encoding are given in [13].

2.4 OpenMath Content Dictionaries

A *Content Dictionary* holds the meanings of (various) mathematical "words" referred to as *symbols*. In the previous release of the *OpenMath* standard, there was a single 'core' CD named `Basic`. In the newest release, a set of official CDs, each covering a specific area has been produced and is available at a public repository site `http://www.nag.co.uk/projects/OpenMath/corecd/`. The `CDGroup MathML` covers essentially the same areas of mathematics as the Content elements of the World Wide Web MathML recommendation [8]. The XML DTD for CDs is given in the *OpenMath* Standard [13].

CDs hold two types of information: that which is pertinent to the whole CD (appears in the header of the CD), and that which is restricted to a particular symbol definition (appears in a CD Definition). Information pertinent to the whole CD includes the name, a description, an expiry date, the status of the CD (official, experimental, private, obsolete), an optional list of CDs on which it depends. Information restricted to a particular symbol includes a name, and a description in natural language. Optional information examples of the use of this symbol, and properties satisfied by this symbol both formal (i.e., in terms of an *OpenMath* object), or commented (i.e., just valid XML). Signatures of symbols are collected in Signature Dictionaries with entries consisting of an *OpenMath* object representing the type of a specific *OpenMath* symbol in a certain type system. Two sets of signature Dictionaries are currently under development: one based on the Calculus of Constructions and its extensions and the other based on an "informal" *OpenMath*-specific type system named STS, for *Simple Type System*.

2.5 OpenMath Phrasebooks

The programs that act as interface between a software application and *OpenMath* are called Phrasebooks. Their task is to translate the *OpenMath* object, as understood using the Content Dictionaries, to the corresponding internal representation used by the specific software application.

Several phrasebooks are under development as part of the *OpenMath* Es-

prit Consortium project. Most notably, prototype versions of phrasebooks for the computer algebra systems AXIOM and GAP are already available. Phrasebooks for Maple[2] and Reduce have been produced in the initial stage of *OpenMath*. For instance, the Maple phrasebook converts Maple notation to the corresponding *OpenMath* abstract object, e.g. the + in the expression $x + y$ is interpreted as an associative and commutative `plus` in Abelian SemiGroups using its definition in the CD `arith1`.

The authors of this paper are currently developing phrasebooks for the proof checkers Lego and Coq as described in more details in Section 4.

Notice that the *OpenMath* phrasebooks are only in charge of translating between *OpenMath* and internal application-specific representation. Control of the interaction among various applications is left to the implementor of the integrated environment. For this task, *OpenMath* allows freedom of choice between several paradigms [9,20].

3 Strong *OpenMath*

Although *OpenMath* does not enforce formally specified signatures of symbols, it recognizes their advantages. In particular, formal signatures in a specific type system can be used to assign mathematical meaning to the object in such a way that validation of *OpenMath* objects depends exclusively on the context determined by the CDs and on some type information carried by the objects themselves. The rest of this paper shows details of how this is achieved in *OpenMath*.

The Calculus of Constructions (CC) and its extensions have been chosen as starting point for assigning signatures to *OpenMath* symbols because they are expressive, well suited to modelling algebra [4,1,26], and have decidable type inference. Various extensions of the Calculus of Constructions have been implemented in systems like Lego or Coq [25,14] that are freely available. These systems can provide the functionalities for performing type checks on *OpenMath* objects. Since the signatures are defined in separate Dictionaries, one can, using the same mechanism outlined in the rest of the paper, choose a different type system and, when possible, convert the available signatures to the new type system.

The approach presented here is to define and use a CD, called `ecc`, for representing terms in an ad-hoc version of ECC [23], *OpenMath*-ECC. In order to be able to assign one ECC-like term to an *OpenMath* object, minor modifications to ECC are needed, *viz.* the introduction of basic constants, and the definition of type universes.

Definition 3.1 (*OpenMath*-ECC terms) *Terms of OpenMath-ECC are defined inductively as follows.*

[2] The version of the Maple phrasebook we used in the example is currently under revision due to recent changes in the CD structure.

(i) *The constants for universes* prop, symtype, *and* omtype, *and the constants* integer, float, string, bytearray *for the types of basic OpenMath objects are terms.*

(ii) *Integers, floating-point numbers, bytearrays, and strings are terms (constants).*

(iii) *Variables* $(x, y, \ldots)$ *are terms.*

(iv) *If* M, N *and* A *are terms, then so are:*

$$MN \quad \lambda x : M.N \quad \Pi x : M.N \quad \Sigma x : M.N \quad \langle M, N \rangle_A \quad \pi_1(M) \quad \pi_2(M).$$

In this definition, as is done in ECC, the pair constructor is heavily typed, it takes the type of the pair A as third argument in order to avoid type ambiguities.

The CD definition of an *OpenMath* symbol can contain a signature corresponding to a type in *OpenMath*-ECC. These signatures are *OpenMath* objects built using the `ecc` CD. Moreover, logical properties of an *OpenMath* symbol can also be formally defined as *OpenMath* objects and can be included as such in the symbol's definitions. *OpenMath* objects that can be mapped to a term in *OpenMath*-ECC are called *Strong OpenMath* objects. The mapping is defined in a natural way.

(i) The *OpenMath* symbols are mapped as described in the CD `ecc`: `integer` to integer, `float` to float, `string` for string, `bytearray` for bytearray, `prop` for prop, `symtype` for symtype, `omtype` for omtype.

OpenMath symbols defined in ecc-CDs (CDs that use `ecc` to assign signatures to the symbols) are *Strong OpenMath* and correspond to constants of appropriate type (as defined in their signature). For instance, the signature of the symbol `posintT` for the type of positive integers can be defined in an ecc-CD as:

```
<signature name="posintT">
              <OMOBJ><OMS cd="ecc" name="symtype"/></OMOBJ>
</signature>
```

and corresponds to a new constant posintT of type symtype.

(ii) Basic objects that are integers, floating-point numbers, bytearrays, and character strings are *Strong OpenMath* and correspond to constants (ground terms) of type integer, float, bytearray, and string, respectively.

(iii) *OpenMath* variables are *Strong OpenMath* and correspond to variables: the *OpenMath* variable with name x corresponds to the *OpenMath*-ECC variable x.

(iv) The remaining *OpenMath* objects are summarized in Figure 1, where $\hat{t}$ in the second column denotes the *OpenMath*-ECC term corresponding to the *Strong OpenMath* object t. The third column concerns the Lego phrasebook translation described in Section 4.

Note that while application is built-in, abstraction and function space

attribution(A,`type` t)	$\hat{A} : \hat{t}$	$\check{A} : \check{t}$
binding(B,**attribution**(v,`type` t), A)	$(\hat{B}\ \hat{t}\ (\lambda\hat{v} : \hat{t}.\hat{A}))$	$(\check{B}\ \check{t}\ [\check{v} : \check{t}]\check{A})$
application(F, A)	$\hat{F}\ \hat{A}$	$(\check{F}\check{A})$
binding(`lambda`,**attribution**(v,`type` t), A)	$\lambda\hat{v} : \hat{t}.\hat{A}$	$[\check{v} : \check{t}]\check{A}$
binding(`PiType`,**attribution**(v,`type` t),u)	$\Pi\hat{v} : \hat{t}.\hat{u}$	$\{\check{v} : \check{t}\}\check{u}$
binding(`SigmaType`,**attribution**(v,`type` t), u)	$\Sigma\hat{v} : \hat{t}.\hat{u}$	$< \check{v} : \check{t} > \check{u}$
application(`Pair`, A_1, A_2, S)	$\langle\hat{A_1}, \hat{A_2}\rangle_{\hat{S}}$	$(\check{A_1}, \check{A_2} : \check{S})$
application(`PairProj1`, A)	$\pi_1(\hat{A})$	$\check{A}.1$
application(`PairProj2`, A)	$\pi_2(\hat{A})$	$\check{A}.2$

Fig. 1. Strong *OpenMath*

type are represented in *OpenMath* by appropriate binding symbols (`lambda`, `PiType`) defined in a new CD called `cc`. In addition, this CD provides the symbols for the names of the types of the basic *OpenMath* objects. The CD Group `ecc` extends the CD `cc` by defining the symbols `Pair` for pairing, `PairProj1` and `PairProj2` for projections and `SigmaType` for the dependent sum type. Notice, in the second line of the above table (in which B is neither `Lambda` nor `PiType` nor `SigmaType`), how new binding symbols can be introduced in CDs: they should be given a signature mapping them to higher-order functions.

An *OpenMath* object in the first column is *Strong OpenMath* if all the subobjects it contains are *Strong OpenMath*. Its semantics is the corresponding *OpenMath*-ECC term in the second column.

Example 3.1 Suppose that the symbolic type RationalT for the type of rationals is introduced in some CD and corresponds to the Sigma type

binding(`SigmaType`,
  attribution(num,`type` integer),
  attribution(den,`type` integer),
  (greater_than den 0)).

Then, the signature for constructor `Rational` is defined by the *Strong OpenMath* object

binding(`PiType`,
  attribution(num,`type` integer),
  attribution(den,`type` integer),
  attribution(rat_cond,`type` (greater_than den 0)),
  RationalT),

This signature represents the term

$$\Pi num : \text{integer}.\Pi den : \text{integer}.\Pi rat\_cond : (\text{greater\_than}\ den\ 0).\text{RationalT}.$$

4 Towards Phrasebooks for Lego and for Coq

Initial work has been done by the authors to interface Lego and Coq to *Strong OpenMath* through a Java client-server applet. The resulting phrasebooks are encoder/decoder Java classes linked with the *OpenMath* Java library provided by the PolyMath Development Group [27].

The Lego phrasebook transforms any *Strong OpenMath* object into the corresponding term in the language used by Lego. Basic objects are translated into new constants of the appropriate type. Column 3 in Figure 1 shows the translation performed by the *OpenMath* Lego phrasebook for compound *Strong OpenMath* objects and $\check{t}$ denotes the Lego term corresponding to the *Strong OpenMath* object t. Figure 2 is a screen-shot of the applet that shows the result of type-checking the expression $\sin(x*y)+x^2+\sin(x)^2$ in a specific Lego context.

As one can see from the figure, the current version of the Lego applet takes as input a mathematical expression in Maple syntax. The Maple phrasebook provided by the *OpenMath* Java library converts this expression into an *OpenMath* object. The Java applet then feeds this object to the Lego phrasebook for obtaining the corresponding Lego term. This Lego term is then shipped to Lego with a context and a command to be performed on the term. Lego runs on a server machine and communicates to the applet the results of the query it has received via the buttons in the graphical user interface. These buttons define which command is sent and will be executed by the server application on the expression given as input. In particular, the query to Lego can be type-checking. If type-checking is successful, then the *Strong OpenMath* object corresponding to the input expression is a meaningful mathematical expression that can be processed further. In the current version of the applet each button is a specific Lego command. We plan to investigate [17,18] among the several possible choices for a more standardized query language.

The Java class implementing the *OpenMath* phrasebook for Lego can of course be re-used for connecting Lego to any other Java application based on the same *OpenMath* library. In fact, we have been able to use the Java phrasebook for the computer algebra package Maple for parsing the input expression and generating an *OpenMath* object. As more phrasebooks for software packages become available, it is easy to imagine an increasingly sophisticated applet that is able to talk to several systems.

In the future we intend to link the applet's functionalities to the IDA project, which brought forth the undergraduate course "Algebra Interactive" [12] used by mathematics and computer science students. In such a project, it is a reasonable prerequisite to check, at the client side, whether the input represents a particular type of mathematical object that is expected for the computation done by the mathematical server (e.g., an integer, a polynomial over a specified ring and with specified indeterminate, or a permutation group). This check should be performed before sending the input to the math-

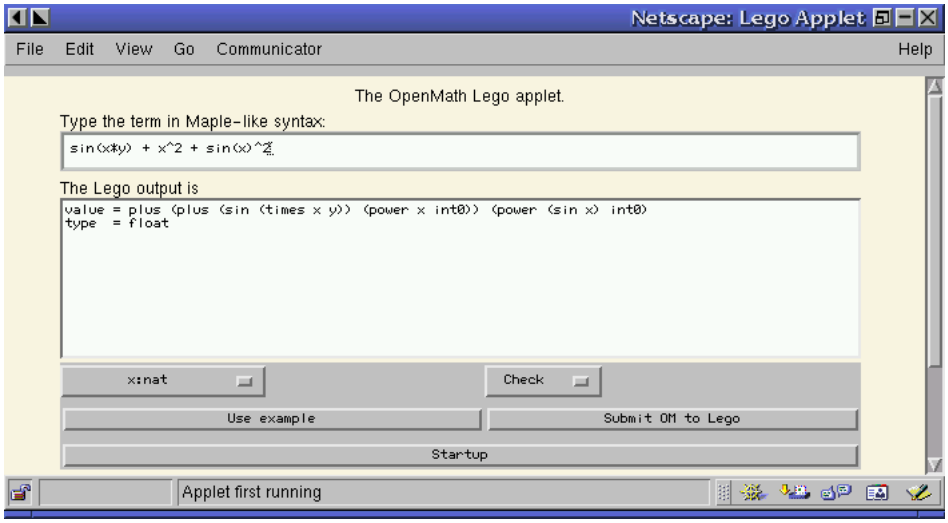

Fig. 2. Screen-shot of the *OpenMath*-Lego Applet

ematical server in order to avoid having to deal with often uncomprehensible error messages. Another direction we intend to investigate is turning the present proofs from static to interactive events. The literature and the state of the art mathematical software [6,24,19] show that more research is needed regarding the correspondence of vernacular with formal mathematics. Along this lines, *OpenMath* needs to be combined with a query language that takes care of the control issues involved in communication among processes. The buttons in the interface of the Java applet partly serve this purpose. It is clear that for a sophisticated integrated environment a more advanced and sound mechanism has to be considered.

5 Conclusion

We have presented the latest version of the *OpenMath* standard. We have also demonstrated by means of a Pure Type System on a substantial sublanguage (called *Strong OpenMath*) of *OpenMath*, that *OpenMath* is suitable for communicating a wide range of mathematics. In fact, in this manner, *OpenMath* can convey proof objects for formal proof checkers as well as mathematical expressions used in computer algebra systems. Thus, *OpenMath* can play a universal role as interface between these two kinds of mathematical computer system and enhance the complementary fuctionalities of several frameworks already present in the literature.

References

[1] *The Lego Algebra Group*. See `http://www.cs.man.ac.uk/~petera/LAG/`.

[2] Iso 7-bit coded character set for information interchange. ISO 646:1983, 1983.

[3] John A. Abbott, André van Leeuwen, and A. Strotmann. *OpenMath*: Communicating Mathematical Information between Co-operating Agents in a Knowledge Network. *Journal of Intelligent Systems*, 1998. Special Issue: "Improving the Design of Intelligent Systems: Outstanding Problems and Some Methods for their Solution.".

[4] Anthony Bailey. *The Machine-Checked Literate Formalisation of Algebra in Type Theory*. PhD thesis, University of Manchester, January 15 January 15th 1998.

[5] C. Ballarin, K. Homann, and J. Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In A.H.M. Levelt, editor, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pages 150–157. ACM Press, 1995.

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

[7] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A Survey of the *Theorema* Project. In *Proceedings of ISSAC'97*, Maui, Hawaii, July 1997. ACM.

[8] Stephen Buswell, Stan Devitt, Angel Diaz, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) 1.0 Specification. W3C Recommendation 19980407, April 1998. Available at `http://www.w3.org/TR/REC-MathML/`.

[9] Piergiorgio Bertoliand Jacques Calmet, Fausto Giunchiglia, and Karsten 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.

[10] P. Chew, R. L. Constable, K. Pingali, S. Vavasis, and R. Zippel. Collaborative mathematics environments. Project Summary.

[11] E. Clarke and X. Zhao. Analytica - a theorem prover in Mathematica. In D. Kapur, editor, *11th Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 761–765. Springer Verlag, 1992.

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

[13] OpenMath Consortium. The OpenMath Standard. OpenMath Deliverable 1.3.2a, February 1999. Available at `http://www.nag.co.uk/projects/OpenMath.html`.

[14] Projet Coq. *The Coq Proof Assistant: The standard library*, version 6.1 edition. Available at `http://www.ens-lyon.fr/LIP/groupes/coq`.

[15] S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 Implementation. pages 241–248. ACM Press, 1997.

[16] Martin Dunstan, Tom Kelsey, Steve Linton, and Ursula Martin. Lightweight Formal Methods for Computer Algebra Systems. In O. Gloor, editor, *ISSAC'98: International Symposium on Symbolic and Algebraic Computation*, Rostock, Germany, August 1998. ACM Press.

[17] Tim Finin, Yannis Labrou, and James Mayfield. Kqml as an agent communication language. In Jeff Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.

[18] Foundation for Intelligent Physical Agents. The fipa '99 baselines. Available at `http://www.fipa.org/spec/fipa99.html`.

[19] A. Franke, S. Hess, Ch. Jung, M. Kohlhase, and V. Sorge. An implementation of distributed mathematical services. In *Calculemus and Types 98*, Eindhoven, July 1998.

[20] 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.

[21] J. Harrison and L. Théry. Reasoning About the Reals: the marriage of HOL and Maple. In A. Voronkov, editor, *Logic Programming and Automated Reasoning: 4th International Conrerence*, volume 698 of *Lecture Notes in Artificial Intelligence*, pages 351–353, St. Petersburg, 1993. Springer-Verlag.

[22] Paul Jackson. *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*. Tr95-1509, Cornell University, 1995.

[23] Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, Edinburgh University, 1990.

[24] 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.

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

[26] L. Pottier and L. Théry. Certifier Computer Algebra. In *Calculemus and Types '98*, Eindhoven, July 1998. `http://www-sop.inria.fr/croap/CFC/`.

[27] PolyMath OpenMath Development Team. Java openmath library, version 0.5. Available at `http://pdg.cecm.sfu.ca/openmath/lib/`, July 1998.