J. Symbolic Computation (2001) **32**, 351–364
doi:10.1006/jsco.2000.0466
Available online at http://www.idealibrary.com on IDEAL®

On the Role of *OpenMath* in Interactive Mathematical Documents

O. CAPROTTI[†] AND A. M. COHEN

Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600MB Eindhoven, The Netherlands

The standard *OpenMath* is an enabling technology for creating an integrated computer environment in which software packages for computer algebra and for proof checking can be combined. Here we demonstrate how *OpenMath* can be employed for generating interactive mathematical documents containing primality proofs. Our case study takes place within a browser; once a prime number is specified, a document appears summarizing the proof in a number of assertions. By clicking an assertion regarding the truth of an arithmetic equality, a computer algebra calculation is invoked verifying the equality. By clicking an assertion regarding a specific mathematical lemma called *Pocklington's Criterion*, a verification of the corresponding formal proof is carried out by a proof checker. Moreover, the whole document is structured in such a way that it can be easily translated to a formal proof object. *OpenMath* supports the interaction between the document as it appears in the browser and the mathematical software packages. This paper begins with an introduction to *OpenMath* and a brief comparison with MathML.

1. Introduction

The observation that computation and deduction are closely related led to several approaches to integrate computer algebra packages with theorem provers. Roughly speaking, there are three directions in which to proceed: include computer algebra capabilities in theorem provers (Harrison and Théry, 1993; Ballarin *et al.*, 1995); include theorem proving capabilities in computer algebra packages (Clarke and Zhao, 1992; Jackson, 1995; Buchberger *et al.*, 1997; Dunstan *et al.*, 1998); and include computing and theorem proving capabilities in a new framework (Chew *et al.*, 1996).

Connections between computer algebra software and proof checker/automated deduction software 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.

The use of *OpenMath* enables a flexible and, indeed, open approach: both the computer algebra packages and the proof checkers are usable via an independent standard language. 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 software package by another.

In this paper we provide an introduction to *OpenMath* (the two authors have been

[†]Supported by the OpenMath Esprit project 24969.

involved in the definition of the *OpenMath* standard language), we demonstrate how to use it for formal mathematics via typing and, finally, we describe a case study in which a primality proof of, say, a 10 digit number can be constructed as an *OpenMath* object and presented in human readable form as an interactive document.

This paper introduces *OpenMath* in Section 2 and compares it to MathML in Section 3. In Section 4 a subclass of *OpenMath* objects, called *Strong OpenMath*, is defined for which meaningfulness cannot only be well defined but also made operational. The case study is presented in Section 5.

2. *OpenMath*

OpenMath is a language for *representing* and *communicating mathematics* (Abbott *et al.*, 1998). Originally, it was conceived as a language for all computer algebra packages (Dalmas *et al.*, 1997; OpenMath, 1999). Currently 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 computer algebra packages.

In addition 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. The great potential of using *OpenMath* technologies to mix computation and deduction is demonstrated by the example of an interactive mathematical document.

OpenMath consists of several aspects. Those presented in this section are: the architecture of how *OpenMath* views integration of computational software, 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 *OpenMath* Standard (OpenMath, 1999) for details.

2.1. *OpenMath* architecture

The architecture of *OpenMath* is made up of three layers of representations of a mathematical object: the private layer for the internal representation, the abstract layer for the representation as an *OpenMath* object, and the communication 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 within 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 initial stage of its 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 such as: integers, symbols, floating-point numbers, character strings, byte arrays, 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.

The integration of computer algebra packages and proof checkers in an interactive document is achieved by means of *OpenMath Phrasebooks*; they convert *OpenMath* objects to and from the software package's internal representation and determine the package's actions. The translation is governed by the CDs and the specifics of the software packages. The example given in this paper uses phrasebooks for COQ and Maple.

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. *OpenMath objects are built recursively as follows.*

(i) Basic *OpenMath* objects, e.g. integers, IEEE floating-point numbers, Unicode character strings, byte arrays, symbols (defined in CDs) and variables, are *OpenMath* objects.

(ii) If $A_1, \dots, A_n (n > 0)$ are *OpenMath* objects, then

$$\mathbf{application}(A_1, \dots, A_n)$$

is an *OpenMath application object.*

(iii) If $S_1, \dots, S_n$ are *OpenMath* symbols, and $A, A_1, \dots, A_n (n > 0)$ are *OpenMath* objects, then

$$\mathbf{attribution}(A, S_1 A_1, \dots, S_n A_n)$$

is an *OpenMath attribution object* and A is the object *stripped of attributions.* The operation of recursively applying stripping to the stripped object is called *flattening of the attribution.* When the stripped object after flattening is a variable, the attributed object is called *attributed variable.*

(iv) If B and C are *OpenMath* objects, and $v_1, \dots, v_n (n \geq 0)$ are *OpenMath* variables or attributed variables, then

$$\mathbf{binding}(B, v_1, \dots, v_n, C)$$

is an *OpenMath binding object.*

(v) If S is an *OpenMath* symbol and $A_1, \dots, A_n (n \geq 0)$ are *OpenMath* objects, then

$$\mathbf{error}(S, A_1, \dots, A_n)$$

is an *OpenMath error object.*

All symbols appearing in an *OpenMath* object are defined in a *Content Dictionary* as described in Section 2.4. In this paper we denote by `foo:bar` the symbol with name `bar` defined in the CD `foo`. Application in *OpenMath* is either intended to be a functional application, like in the object **application**(`transc1:sin`, x), or a constructor, like in **application**(`foo:Rational`, 1, 2). Binding objects, like **binding**(`lc:Lambda`, x, **application**(`arith1:plus`, x, 2)), can be used to express logical statements. Attribution can act as either "annotation", in the sense of adornment, or as "modifier". Attribution is used in Section 4 to express the judgment that says that object A has type t by **attribution**(A, `ecc:type` t). Errors occur during the manipulation of *OpenMath* objects and so are of real interest only when communication is taking place.

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 (ISO, 1983) (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 **application**(`rem`, **application**(`gcd`, 12, 3), 2) is:

```
<OMOBJ>
    <OMA><OMS cd="integer" name="rem"/>
         <OMA><OMS cd="integer" name="gcd" />
              <OMI>12</OMI>
              <OMI>3</OMI>
         </OMA>
         <OMI>2</OMI>
    </OMA>
</OMOBJ>
```

Here, `OMOBJ` encapsulates an *OpenMath* object, `OMA` indicates an **application** object, `OMS` a symbol defined by the `name` and `cd` elements and `OMI` an integer.

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.

2.4. *OpenMath* CONTENT DICTIONARIES

A *Content Dictionary* holds the meanings of (various) mathematical 'words' referred to as *symbols*. A set of official CDs, each covering a specific area has been produced and is available from the CD repository of the *OpenMath* society. CDs may be grouped into *CD groups*, so that applications can easily refer to collections of CDs. For instance, the `MathML` CD group covers essentially the same areas of mathematics as the content elements of the Web Consortium MathML recommendation (Buswell *et al.*, 1998). The `Types` CD group used in Section 4 collects symbols used in languages for type theory.

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 expiration date, the status of the CD (official, experimental, private, obsolete), and 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 formal properties satisfied by this symbol (i.e. theorems expressed *OpenMath* objects), or commented (i.e. just valid XML).

Formal signatures and definitions are collected in additional files that can be associated to a CD. Axiomatic definitions in a specific formal system are expressed as *OpenMath* objects in *DefMP* (defining mathematical property) files. Signatures of symbols are expressed in *Signature* files by *OpenMath* objects representing types in a certain type system. We rely on this in Section 4 where we give a detailed description of how to assign formal signatures to *OpenMath* objects, whereas in Section 5 we give an example of a defining mathematical property for an *OpenMath* symbol. A set of these files based on extensions of the Calculus of Constructions is also available from the CD repository.

2.5. *OpenMath* PHRASEBOOKS

The programs that act as an interface between a software system and *OpenMath* are called *phrasebooks.* Their task is to translate the *OpenMath* object, as understood by means of CDs, to the corresponding internal representation used by the specific software package.

Several phrasebooks are under development as part of the *OpenMath* Esprit Consortium project. Most notably, prototype versions of phrasebooks for the computer algebra packages AXIOM and GAP are already available. These are examples in which the software package itself provides the interface to and from *OpenMath* internally.

Phrasebooks for Maple and REDUCE have been produced in the initial stage of *OpenMath.* The current release of the Naomi *OpenMath* JAVA Library by the PolyMath Development Group (PolyMath, 1999) also includes, besides classes representing *OpenMath* objects, phrasebook classes. In particular, it is possible to encode and decode arithmetical expressions written in Mathematica or in Maple syntax by using the respective phrasebooks. This approach is independent of the specific software package, which is left untouched. For instance, the Maple phrasebook converts Maple notation to the corresponding *OpenMath* abstract object, e.g. the $+$ in the expression $x + y$ is represented by use of `arith1:plus`, which is associative and commutative addition in Abelian semigroups according to the definition in the CD `arith1`. A Mathematica phrasebook based on the *OpenMath* C library is available at INRIA.

In our development, we have taken the approach of not modifying the software packages we want to interface. Therefore, we independently built phrasebooks for the proof checkers Lego and COQ using the Naomi Java library.

Note that the *OpenMath* phrasebooks are mainly concerned with the translation between *OpenMath* objects and internal system-specific representations. The interpreted behavior for most computer algebra packages receiving an *OpenMath* object is an evaluation. For instance, upon input of the object **application**(`integer:rem`, **application**(`integer:gcd`, 12, 3), 2) Mathematica would output 1.

In general, control of the interaction with a software package is not ruled by the existing *OpenMath* libraries. For this task, *OpenMath* allows freedom of choice between several paradigms (Bertoli *et al.*, 1998; Franke *et al.*, 1999).

3. *OpenMath* and MathML

In the previous section we introduced *OpenMath*. Because of its XML encoding, *OpenMath* may seem to be a mark-up language for mathematics that is competing with MathML (Caprotti and Carlisle, 1999; Buswell *et al.*, 2000). In fact, the two languages complement each other: MathML can be used for presenting mathematical content written in *OpenMath*, especially in areas where MathML-Content does not suffice; the type theoretical concepts introduced in Section 4 are examples of this. Moreover, *OpenMath* is designed to be extensible, whereas MathML-Content covers K–12 mathematics and *de facto* refers to *OpenMath* for extensions.

Translation tools (Buswell, 1999) have been produced for the common subset of mathematics covered by MathML-Content and by the *OpenMath* `MathML` CD group. For presenting *OpenMath* objects by use of MathML-Presentation, one may use XSL stylesheets driven by *OpenMath* CDs. Sample stylesheets are available from the *OpenMath* Society web pages.

The generality of *OpenMath* makes it difficult for general-purpose software to implement it so that it captures all the functions available; the package's phrasebook would need a great variety of CDs. In this vein it is to be expected that specialized packages will interface to *OpenMath*, leaving MathML to the less specific software packages. But another development may be that general-purpose packages are supplied with several phrasebooks, one for each (specific) purpose. For instance, in order to use Maple or Mathematica as a back engine to an interactive document such as *Algebra Interactive!* (Cohen *et al.*, 1999), a phrasebook supporting a very limited number of CDs would be needed.

4. Strong *OpenMath*

Although *OpenMath* does not enforce formal specification of symbols by using signatures and axiomatic definitions, it recognizes their advantages and leaves room to adopt them if required. In particular, formal signatures in a specific type system can be used to assign mathematical meaning to *OpenMath* objects in such a way that validation of objects depends exclusively on the context determined by the CDs and on some type information carried by the objects themselves. When defining mathematical properties (introduced in Section 2.4) are used to axiomatize symbols in terms of previously introduced symbols, automated verification techniques can be applied. In this section, we give details on how this is achieved.

Extensions of the Calculus of Constructions (CC) have been chosen as a starting point for assigning signatures to *OpenMath* symbols because they are expressive, well suited to modeling algebra (Bailey, 1998; Pottier and Théry, 1998; LEGO, 1999), and have decidable type inference. Various extensions of the CC have been implemented in freely available software packages such as Lego or COQ (Luo and Pollack, 1992; Coq, 1999). These packages provide the necessary functionality for performing type checks on *OpenMath* objects. Since the signatures are defined in separate Signature files, it is easy to switch type systems: simply convert, wherever applicable, the available signatures to the new type system.

To exemplify how formal meaning can be assigned to *OpenMath* objects, we take an instance of the formal type system called Extended Calculus of Constructions (ECC) (Luo, 1990) as semantics for *OpenMath*. We call it OM-ECC; it has a shallow hierarchy of type universes for the basic *OpenMath* objects. A specific universe is assigned to the type of *OpenMath* types.

DEFINITION. (OM-ECC TERMS) Terms of OM-ECC are defined inductively as follows.

(1) The constants for universes Prop, symtype, and omtype, and the constants integer, float, string, bytearray for the types of basic *OpenMath* objects are terms.
(2) Integers, floating-point numbers, strings, and bytearrays are terms (constants).
(3) Variables $(x, y, \dots)$ are terms.
(4) 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. In order to avoid type ambiguities, we take the type A of the pair as its third argument.

Now we consider how to employ *OpenMath* for expressing OM-ECC. Whereas application is built-in, *OpenMath* has to be equipped with new binding symbols for constructing types and with constants expressing type universes. The Types CD group collects all CDs that define symbols for expressing formal terms and types. In particular, abstraction and function space type are represented using binding symbols `lc:Lambda` (λ), `lc:PiType` (Π) defined in a CD called `lc` (for lambda calculus). The CD `omtypes` provides the symbols for the names of the types of the basic *OpenMath* objects. The CD `typesorts` defines the symbol `typesorts:Prop` for the universe of propositions. The symbols `ecc:Pair` ($\langle\rangle$) for pairing, `ecc:PairProj1` and `ecc:PairProj2` (π_1 and π_2) for projections, and `ecc:SigmaType` (Σ) for the dependent sum type are defined in the CD `ecc`.

Now that we have introduced *OpenMath* symbols to represent the language of OM-ECC, we assign the formal meaning to a class of *OpenMath* objects in terms of OM-ECC. As a consequence, for this class of objects, the meaning of the *OpenMath* constructor **application** will depend on the head function symbol: provided the head is not the symbol `ecc:Pair`, it will be a functional application as intended in lambda calculus. The *OpenMath* constructor **binding** will take its semantics from the binding symbol.

Below we define a natural (partial) semantics for *OpenMath* by OM-ECC. *OpenMath* objects that can be mapped by the informal procedure described here to a term in OM-ECC are called *Strong OpenMath* objects.

Basic Objects The *OpenMath* symbols in the CD `omtypes` represent OM-ECC type constants; more precisely, `omtypes:integer` for integer, `omtypes:float` for float, `omtypes:string` for string, `omtypes:bytearray` for bytearray, `omtypes:symtype` for symtype, and `omtypes:omtype` for omtype. The symbol `typesorts:Prop` is interpreted as the constant for the universe Prop of propositions.

An *OpenMath* symbol defined in a CD is mapped to a constant in OM-ECC only if the following condition is satisfied. The symbol definition in the CD is linked either to a formal signature or to a formal defining mathematical property. These are given in terms of *OpenMath* objects that map to OM-ECC. When this is true, then the *OpenMath* symbol is mapped to a constant in OM-ECC of the appropriate type. For instance, if the symbol `foo:posintT` for the type of positive integers is linked to the signature `omtype:symtype`, then it corresponds to a new OM-ECC constant posintT of type symtype.

Table 1. Strong *OpenMath*.

OpenMath	OM-ECC
attribution(A,`ecc:type` $t, \dots$)	$\hat{A} : \hat{t}$
attribution($A, S\, t, \dots$)	$\hat{A}$
binding(`lc:Lambda`,**attribution**(v, `ecc:type` t), A)	$\lambda\hat{v} : \hat{t}.\hat{A}$
binding(`lc:PiType`,**attribution**(v, `ecc:type` t),u)	$\Pi\hat{v} : \hat{t}.\hat{u}$
binding(`ecc:SigmaType`,**attribution**(v ,`ecc:type` t), u)	$\Sigma\hat{v} : \hat{t}.\hat{u}$
binding(B,**attribution**(v,`ecc:type` t), A)	$(\hat{B}\ \hat{t}\ (\lambda\hat{v} : \hat{t}.\hat{A}))$
application(`ecc:Pair`, A_1, A_2, S)	$\langle\hat{A}_1, \hat{A}_2\rangle_{\hat{S}}$
application(`ecc:PairProj1`, A)	$\pi_1(\hat{A})$
application(`ecc:PairProj2`, A)	$\pi_2(\hat{A})$
application(F, A)	$\hat{F}\ \hat{A}$

The basic objects, integers, floating-point numbers, character strings, and byte arrays, are *Strong OpenMath* and correspond to constants (OM-ECC ground terms) of type integer, float, string, and bytearray, respectively.

OpenMath variables are *Strong OpenMath* and correspond to variables: more precisely the *OpenMath* variable with name x corresponds to the OM-ECC variable x.

Compound Objects The remaining *OpenMath* objects are summarized in Table 1, where $\hat{t}$ in the second column denotes the OM-ECC term corresponding to the *Strong OpenMath* object t. The interpretation proceeds top down from the root of the *OpenMath* tree down to the basic objects in the leaves.

A few remarks are in order. An attributed object is interpreted as the stripped object (as in the second line of Table 1 where S is not the symbol `ecc:type`) except when one of the attributes is the symbol `ecc:type`. In this case the attributed object is interpreted as a type judgment.

The sixth line of Table 1 (in which B is neither `lc:Lambda` nor `lc:PiType` nor `ecc:SigmaType`) states that new binding symbols can be introduced in CDs provided they are given a signature mapping them to higher-order functions. A binding object of the form **binding**$(B, v_1, \dots, v_n, A)$ is considered an abbreviation of **binding**$(B, v_1,$ **binding**$(B, v_2, \dots))$. Similarly, an application object of the form **application**$(F, A_1, \dots, A_n)$ (where F is not the symbol `ecc:Pair`) is considered an abbreviation of the object **application**$(\dots((F, A_1), A_2)\dots, A_n)$.

OpenMath error objects are not given any formal meaning because, in this setting, we do not consider them to be mathematical objects.

An *OpenMath* object in the first column of Table 1 is *Strong OpenMath* if all the subobjects it contains are *Strong OpenMath*. Its semantics is the corresponding OM-ECC term recursively obtained in the second column. In other words, the class of *OpenMath* objects for which the mapping above is well defined is called *Strong OpenMath*. The advantage of *Strong OpenMath* is that, for this class of objects, the notion of meaningfulness coincides with that of having a type. Moreover, it can be decided whether an *OpenMath* object is meaningful by means of an efficient algorithm, for instance the type checking algorithm implemented in the package COQ (or Lego).

5. A Specimen Interactive Mathematical Document

One of the major application areas for *OpenMath* is envisioned to be that of publishing interactive mathematics. In this section, we illustrate by means of a case study some of the interactivity that can already be achieved in a browser. To this end, we bring together the software packages Coq, GAP and Maple.

Researchers in formal mathematics stress the importance that a mathematical text be checked automatically for its validity and correctness on the basis of some initial assumptions. Projects such as Automath (Nederpelt *et al.*, 1994) and Mizar (Trybulec, 1980) confirm this point of view. On the other hand, texts that are actually machine checkable do not appear palatable to humans.

Several partial solutions to the problem of producing text which is on the one hand fit for human reading and on the other hand automatically verifiable have been offered. For instance, the editing tool of Coq demonstrates how the mathematical vernacular can dress up each step used in a formal proof and transform it to human-readable jargon. The advantage is clearly that the computer acts as a kind of referee for the logical correctness of the results. However, because each single assertion is checked, even elementary theorems of first-year algebra become quite cumbersome to prove. In general, proofs of theorems in algebra require arithmetic, which might be carried out by a computer algebra package, and the logic flow of inference amongst steps in the proof, which might be relegated to an automated theorem prover.

In order to apply this idea to a representative case in point, we considered the primality proof of an integer n by use of Pocklington's Criterion. We shall first recall the criterion, next its application in producing an interactive proof of the primality of a user-input number and, finally, describe a formalization in Coq.

The criterion has already been studied from a formal point of view by Elbers (1998).

Lemma 5.1. (Pocklington's Criterion) *Let* $n \in \mathbb{N}$, $n > 1$ *with* $n-1 = q \cdot m$ *such that* $q = q_1 \cdots q_t$ *for certain primes* $q_1, \ldots, q_t$. *Suppose that* $a \in \mathbb{Z}$ *satisfies* $a^{n-1} = 1 \pmod{n}$ *and* $\gcd(a^{\frac{n-1}{q_i}} - 1, n) = 1$ *for all* $i = 1, \ldots, t$. *If* $q \geq \sqrt{n}$, *then* n *is a prime.*

A theorem prover like Coq can use the criterion for showing primality of a certain number n if it knows the numbers q, $q_1, \ldots, q_t$, a (and recursively so for the primes $q_1, \ldots, q_t$) and if it can deal with ordinary and modular arithmetic, including gcd's. Given these witnesses, showing that n is prime amounts to verifying that all conditions of Pocklington's Criterion are met. Most of these conditions are trivially verified by a computer algebra package, but they are also relatively simple for a proof checker. Proving that the numbers $q_1, \ldots, q_t$ are prime is done by recursive calls of Pocklington's Criterion except when the prime number is 2.

In the computer algebra package GAP it is easy to write a program that supplies the requested witnesses to a human or to a theorem prover; for instance, a is given by GAP's function `PrimitiveRootMod()` which produces a primitive root a modulo n; that is, an element a such that $a^{n-1} = 1 \pmod{n}$ and $a^i \neq 1 \pmod{n}$ for $i = 1, \ldots, n-2$.

In our setup, we provide a web page in which the user types a positive number n as a result of which either a factorization of n is returned or a proof of primality of n is produced in which each assertion roughly corresponds to a subgoal in the formal proof. The program that produces this trace of the primality proof is written in GAP and recursively produces the trace for all the prime numbers needed in the initial proof.

The returned document[†] is interactive as it is possible to open and close proofs, and to validate every arithmetic assertion using a computer algebra package. This is achieved by enabling calls to a Java applet which controls the interaction between the browser and converts the available mathematical services. It also acts as a phrasebook for the packages providing these services. For example, for validation of a mathematical object, the applet sends out a command such as `evalb` for Maple and `Check` for Coq, applied to the properly translated object, and translates the output of the package back into *OpenMath*. Because of the use of *OpenMath* and of the customizability of the Java applet, the requested services can be provided over the net by a wide range of mathematical software packages.

The document returned by GAP is written in OMDoc, an extension of *OpenMath* for including *OpenMath* objects in mathematical documents (Kohlhase, 2000). Note that *OpenMath* alone is not enough to express a corpus of mathematical knowledge because it lacks the mechanisms to relate such concepts as definitions, theories, and proofs of theorems. Moreover, the command language used during an interactive session with a computational tool is not cleanly specified within *OpenMath*. These limitations are the reasons why *OpenMath* documents have been introduced. Currently, OMDoc is being used as a source format for the next release of *Algebra Interactive!* (Cohen *et al.*, 1999), and for the knowledge base of mathematics MBase (Kohlhase and Franke, 2001). We will not enter the details of OMDoc here but concentrate on the *OpenMath* aspects.

Included here is a textual copy of the initial assertions in the document produced interactively for proving primality of the 10-digit number 1234567891:

```
Assertion 1: Pocklington's Criterion.
Assertion 2: The number 1234567891 is prime because of
 * Pocklington's Criterion applied with n = 1234567891,
   q = 13717421, m = 90, t = 2, q[1] = 3803, q[2] = 3607.
 * Assertions 3,..., 8, verifying Pocklington's conditions,
 * Primality of the numbers 3803, 3607, which will be
   established below, see Assertions 9, 15.

Assertion 3: 1234567890 = 13717421 * 90
Assertion 4: 13717421 = 3803*3607
Assertion 5: 13717421^2 >= 1234567891
Assertion 6: 3^(1234567891-1) = 1(mod 1234567891)
Assertion 7: gcd(3^((1234567891-1)/3803)-1,1234567891) = 1
Assertion 8: gcd(3^((1234567891-1)/3607)-1,1234567891) = 1
Assertion 9:The number 3803 is prime because of
 * Pocklington's Criterion applied with n = 3803, q = 1901,
   m = 2, t = 1, q[1] = 1901.
 * Assertions 10,..., 14, verifying Pocklington's conditions,
 * Primality of the number 1901, which will be established
   below, see Assertion 21.
```

The total number of assertions in the document is, in this case, 52. These assertions are represented as *OpenMath* objects in the source using the CDs `arith1`, `relation1`,

[†]`http://crystal.win.tue.nl/~olga/openmath/pocklington/`.

`integer`, `setname` and the CDs for the types `omtypes`, `lc`, `cc`, and `icc`. On screen they are rendered in human-readable syntax by a conversion of the XML source (in OMDOC format) to HTML.

In order to include a formalization for the statement of Pocklington's Criterion itself, we take the *OpenMath* object that represents the formal definition in COQ described below. Again, we have to equip *OpenMath* with the right symbols to be able to express the notions involved in stating the criterion. As a guideline, we take the symbols used in the COQ context. This is done by introducing a private CD, called `pock`, and by giving, besides a natural language description, a formal definition in terms of a `DefMP`. For instance, the *OpenMath* symbol `pock:divides` is defined by the property:

binding(`lc:Lambda`,
  attribution(n, `icc:type`, `setname:N`),
  attribution(m, `icc:type`, `setname:N`),
  binding(`quant1:exists`,
    attribution(q, `icc:type`, `setname:N`),
    application(`relation1:eq`, m,
    application(`arith1:times`, `setname:N`, n, q))))

When feeding this object to COQ, our COQ phrasebook translates it to the COQ definition of the symbol `divides` thus extending COQ's context.

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

Note that the phrasebook translates the *OpenMath* symbols `relation1:eq` and `arith1:times`, available in official CDs, to COQ's library functions `=` and `mult`. The formal signatures of the symbol `arith1:times`, specifying the domain on which multiplication lives, disambiguates the translation of **application**(`arith1:times`, `setname1:N`, n, q) to `(mult n q)`, where `mult` is COQ's multiplication over the natural numbers. Strictly speaking, because we are using COQ, the type system considered for this example is the Inductive Calculus of Constructions. The additional constructors for inductive types are available as *OpenMath* symbols in the CD `icc`. Once all the symbols needed are given in *OpenMath*, it is an easy exercise to construct the *OpenMath* object stating Pocklington's Criterion.

Finally, we briefly describe a COQ formalization of Pocklington's Criterion that is indeed very close structurally to the *OpenMath* object and has the added advantage of being more concise. We assume that the context here contains notions available from the COQ standard libraries and notions that are user defined, i.e. for the type of lists of natural numbers (`natlist`), for the product of all elements in a list (`product`), and for testing that a predicate holds on all members of a list (`allPrime`, `allLinComb`). Then Pocklington's Criterion is formally stated as follows.

```
Theorem pocklington:
   (n,q,m:nat)(a:Z)(qlist:natlist)
      (gt n (1)) ->
         n=(S (mult q m)) ->
```

```
q=(product qlist) ->
   (allPrime qlist) ->
      (Mod (Exp a (pred n)) ‘1’ n) ->
         (allLinComb a n m qlist) ->
            (le n (mult q q)) ->
               (prime n).
```

We have started to investigate the possibility of producing and including in the source document the *OpenMath* object representing the formal proof object, of type say **application**(`pock:prime`,1234567891). A JAVA applet, jointly developed in collaboration with Oostdijk (Caprotti and Oostdijk, 2000), generates a COQ tactic script for proving primality of an arbitrary number n (practically, the number n is bound by the size of the maximum natural number available in COQ). The script is then used by COQ to produce the proof object which is representable as an *OpenMath* object. Again, the extensionality of *OpenMath* allows us to define a private `coq` CD for symbols that correspond to inference rules appearing in COQ's proof objects and are formally defined in terms of the inductive constructor symbols given in the CD `icc`.

6. Conclusion

Now that the *OpenMath* Standard has come about, the question arises of how to structure documents containing mathematics represented by *OpenMath* objects. Such a document could be a single *OpenMath* object, for instance an *OpenMath* sequence of *OpenMath* objects, many of which are strings representing text. Another approach would be to allow for *OpenMath* objects appearing in an XML document as was done in so-called "*OpenMath* documents". An advantage of the latter approach might be that it allows for greater interactivity. In a case study in this direction, we have written an interactive document that, upon inputing a number, decides whether or not it is prime (if not too big, of course), and if so, delivers a proof in words, containing assertions about numbers. Since the arithmetic statements are produced in *OpenMath*, they can be checked by any software package that understands a specific, small set of *OpenMath* CDs. Likewise, since the reasoning is by standard (natural language) sentences, it can easily be translated into a formal proof, checkable by, say, COQ, perhaps modulo the arithmetic. The interactive production of the document containing the proof is carried out by a GAP program running in the background.

Although *OpenMath* is suitable for describing mathematical objects in a context defined by means of CDs, Signature files and DefMP files; so far, little attention has been paid to the inter-relations among these definitions and (defining) properties. Our case study shows that we can set up a context adequate for defining a rigorous primality proof that is both human-readable, and verifiable by COQ and Maple, or by GAP and Lego. In the document and in the interaction with these background software packages, *OpenMath* is the common language.

The *OpenMath* document produced in our case study demonstrates the power of blending informal, formal and computational aspects into one entity. By perusing, clicking, and elaborating the document, the reader may convince themself of the correctness (validity) of the content. We envisige that this is the way a lot of mathematics will be handled in the future.

Acknowledgements

The authors wish to thank the anonymous referees for the insightful comments on the first draft of this article. A special thank goes to all members of the *OpenMath* Society for the fruitful cooperation during the past years that finally led to the definition of the *OpenMath* standard language.

References

Abbott, J. A., van Leeuwen, A. M., Strotmann, A. (1998). *OpenMath*: Communicating mathematical information between co-operating agents in a knowledge network. *J. Intell. Syst.* Special Issue: Improving the Design of Intelligent Systems: Outstanding Problems and Some Methods for their Solution. `http://www.brunel.ac.uk/~hssrjis/issue/index8.html`.

Bailey, A. (January 15th 1998). The machine-checked literate formalisation of algebra in type theory. Ph.D. Thesis, University of Manchester.

Ballarin, C., Homann, K., Calmet, J. (1995). Theorems and algorithms: an interface between Isabelle and Maple. In Levelt, A. ed., *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, Montreal, Canada, pp. 150–157. New York, ACM Press.

Bertoli, P., Calmet, J., Giunchiglia, F., Homann, K. (1998). Specification and integration of theorem provers and computer algebra systems. In Calmet, J. , Plaza, J. eds, *Artificial Intelligence and Symbolic Computation: International Conference AISC'98, Plattsburgh, New York, U.S.A.*, LNAI **1476**.

Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuta, E., Vasaru, D. (1997). A survey of the *Theorema* project. In *Proceedings of ISSAC'97, Maui, Hawaii*. ACM.

Buswell, S. (1999). STARS. Stilo Technologies. Available online at `http://www.stilo.com`.

Buswell, S., *et al.* (2000). Mathematical Markup Language (MathML) Version 2.0. W3C Working Draft 11 February 2000. Available online at `http://www.w3.org/TR/2000/WD-MathML2-20000211/`.

Buswell, S., Devitt, S., Diaz, A., Poppelier, N., Smith, B., Soiffer, N., Sutor, R., Watt, S. (1998). Mathematical Markup Language (MathML) 1.0 Specification. W3C Recommendation 19980407. Available online at `http://www.w3.org/TR/REC-MathML/`.

Caprotti, O., Carlisle, D. (1999). OpenMath and MathML: Semantic mark up for mathematics. *Crossroad*, Special Issue on Markup Languages. Available online at `http://www.acm.org/crossroads`.

Caprotti, O., Oostdijk, M. (July 2000). Proofs in interactive mathematical documents. In *Proceedings of AISC 2000, Artificial Intelligence and Symbolic Computation, Theory, Implementations and Applications, Madrid, Spain*, pp. 17–19.

Chew, P., Constable, R. L., Pingali, K., Vavasis, S., Zippel, R. (1996). Collaborative mathematics environments. Project Summary.

Clarke, E., Zhao, X. (1992). Analytica—a theorem prover in mathematica. In Kapur, D. ed., *11th Conference on Automated Deduction*, LNCS **607**, pp. 761–765. New York, Springer Verlag.

Cohen, A. M., Cuypers, H., Sterk, H. (1999). *Algebra Interactive, Interactive Course Material.* Number ISBN 3-540-65368-6. SV.

Coq, P. (1999). *The Coq Proof Assistant: The Standard Library*, Version 6.3.1 edn, Available online at `http://www.ens-lyon.fr/LIP/groupes/coq`.

Dalmas, S., Gaëtano, M., Watt, S. (1997). *An OpenMath 1.0 Implementation.* In *Proceedings of ISSAC' 97, Mavi, Hawaii*, pp. 241–248. New York, ACM Press.

Dunstan, M., Kelsey, T., Linton, S., Martin, U. (1998). Lightweight formal methods for computer algebra systems. In Gloor, O. ed., *ISSAC'98: International Symposium on Symbolic and Algebraic Computation, Rostock, Germany.* New York, ACM Press.

Elbers, H. (1998). Connecting informal and formal mathematics. Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Franke, A., Hess, S., Jung, C., Kohlhase, M., Sorge, V. (1999). Agent-oriented integration of distributed mathematical services. *J. Universal Comput. Sci.*, **5**, 156–187. Special issue on Integration of Deduction System.

Harrison, J., Théry, L. (1993). Reasoning about the reals: the marriage of HOL and Maple. In Voronkov, A. ed., *Logic Programming and Automated Reasoning: 4th International Conference, St Petersburg*, LNAI **698**, pp. 351–353. Berlin, Springer-Verlag.

ISO (1983). *ISO 7-bit Coded Character Set for Information Interchange.* `http://www.iso.ch/iso/en/Standards_Search`. Standards Query Form.

Jackson, P. (1995). *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra.* Tr95-1509, Cornell University.

Kohlhase, M. (2000). OMDOC: Towards an OPENMATH representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes. Available online at `http://www.mathweb.org/ilo/omdoc`.

Kohlhase, M., Franke, A. (2001). MBase: Representing knowledge and context for the integration of mathematical software systems. *J. Symb. Comput.*, **32**, 365–402.

LEGO (1999). *The Lego Algebra Group*. Available online at `http://www.cs.man.ac.uk/~petera/LAG/`.

Luo, Z. (1990). An extended calculus of constructions. Ph.D. Thesis, Edinburgh University.

Luo, Z., Pollack, R. (1992). *LEGO Proof Development System: User's Manual*. University of Edinburgh, Department of Computer Science.

Nederpelt, R. P., Geuvers, J. H., de Vrijer, R. C. eds. (1994). *Selected Papers on Automath*. North-Holland, Amsterdam.

OpenMath, C. (1999). The OpenMath Standard. OpenMath Deliverable 1.3.3a. In Caprotti, O., Carlisle, D. P., Cohen, A. M. eds, OpenMath Esprit Consortium. Available online at `http://www.nag.co.uk/projects/OpenMath.html`.

PolyMath (1999). Java OpenMath Library: Version 0.7c.

Pottier, L., Théry, L. (1998). Certifier computer algebra. In *Calculemus and Types '98*. Eindhoven. Available online at `http://www-sop.inria.fr/croap/CFC/`.

Trybulec, A. (1980). The Mizar Logic information language. *Stud. Logic, Grammar Rhetoric*, **1**.

Originally Received 8 November 1999
Revised 10 April 2000
Accepted 1 November 2000