XML, Stylesheets and the re-mathematization of Formal Content

Andrea Asperti Luca Padovani Claudio Sacerdoti Coen
Irene Schena

March 30, 2001

Abstract

An important part of the descriptive power of mathematics derives from its ability to represent formal concepts in a highly evolved, two-dimensional system of symbolic notations. Tools for the mechanisation of mathematics and the automation of formal reasoning must eventually face the problem of re-mathematization of the logical, symbolic content of the information, especially in view of their integration with the World Wide Web. In a different work [3], we already discussed the pivotal role that **XML** (eXtensible Markup Language) technology [8] is likely to play in such an integration. In this paper, we focus on the problem of (Web) publishing, advocating the use of **XSL** (eXtensible Stylesheet Language) Transformations, in conjunction with **MathML** (Mathematical Markup Language), as a standard, application independent and modular way for associating notation to formal mathematical content.

1 Introduction

In a different paper [3] we advocated the pivotal role of **XML**-technology, recently introduced by the **W3C** (World Wide Web Consortium), for the development of large, distributed repositories of formal mathematical knowledge. The eXtensible Markup Language [8], which is rapidly imposing as the main tool for representation, manipulation, and exchange of structured information in the networked age, provides a simple bridge between tools for automation of formal reasoning and the Web, and naturally leads to the design of a new generation of Logical Environments along the guidelines of the new Information Society, and its emphasis on *content*. The broad goal of our project (`http://www.cs.unibo.it/helm`) goes far beyond the trivial suggestion to adopt **XML** as a neutral specification language for the 8221;compiled” versions of the libraries, or even the observation that in this way we could take advantage of a lot of functionalities on **XML**-documents already offered by standard commercial tools. First of all, having a common, application independent, meta-language for mathematical proofs, similar software tools could be applied to different logical dialects, regardless of their concrete nature. This would be especially relevant for all those operations like rendering, browsing, searching, and retrieving (just to mention a few of them) that are largely independent from the specific logical system. Moreover, if having a common representation layer is not the ultimate solution to all interoperability problems between different applications, it is however a first and essential step in this direction. Finally, this 8221;standardisation” process naturally leads to a substantial simplification and re-organisation of the current, 8221;monolithic” architecture of logical frameworks. All the many different and often loosely connected functionalities of these complex programs (proof checking, proof editing, proof displaying, search and consulting, program extraction, and so on) could be clearly split in more or less autonomous tasks, possibly (and hopefully!) developed by different teams, in totally different languages. This is the new, 8221;content-centric" architectural design of future systems.

An essential component of **XML**-technology is **XSLT** (Extensible Stylesheet Language, Transformation part) [10] which recently became a **W3C** Recommendation. **XSLT** is a simple rule-based language for transforming **XML** documents into other **XML** documents. Although the expressive power of **XSLT** is remarkable, it was not intended as a completely general-purpose **XML** transformation language but, primarily, as a mean to specify the *styling* of an **XML** document by transforming the specific **XML**-dialect of the input document into a formatting language suitable for rendering issues (**HTML** (HyperText Markup Language), Formatting Objects, or whatever). This is also the use of **XSLT** intended in this paper; in particular, we advocate the use of stylesheets as a standard mechanism for associating notation to content in mathematical documents.

This raises the question of the target formatting language for mathematics. Once in the realm of **XML**, a natural choice is provided by **MathML**[1]. **MathML** [11] is an instance of **XML** which has been developed under the aegis of **W3C** with the goal to enable mathematics to be served, received, and processed on the Web, just as **HTML** has enabled this functionality for text. **MathML** is actually composed by two parts: *presentation* markup (which should become, roughly, the math-mode of **HTML**), and *content* markup, whose aim should be to capture the 8221;logical structure" of the information. **MathML** also comprises some mechanisms for associating content with presentation. While **MathML**-presentation is well on its way and likely to be rapidly adopted by several browsers, the actual role of **MathML**-content is more questionable, especially for people aware of (and accostumed to) the multi-lingual environment of the foundations of mathematics. Still, **MathML**-content has been conceived as an extensible (and thus, potentially, infinitary) language; this fact, and its flexible nature, make it very suitable to act as an interesting intermediate semi-formal language between the specific (logic-dependent) low level representation of the information and its presentation in some editing format. The fact of passing through an intermediate representation is going to improve the modularity of the overall architecture: many specific logical dialects can be mapped into the same intermediate language (or into suitable extensions of it), and many transformations to specific editing formats can be defined for the single intermediate language (just simple extensions of the transformations would be required, in case of extensions of the intermediate representation).

2 Proofs as expressions

Accordingly to the Semantic Web, we are interested not only in encoding mathematical documents, and proofs in particular, in a machine readable way, but also in a machine understandable way. This means, for example, that it must be easy to write tools to inspect proofs, to automatically verify their correctness and to data-mine them. This has always been the subject of study of Proof Theory and Formal Logic, whose concrete products are Proof Assistants.

In almost every Proof Assistant, a formal proof is encoded in a very peculiar way, which is an expression in a particular calculus. Even a short introduction to the subject would require a whole text-book; the best reference for beginners is probably [6]. Thus, in the rest of this section we will only show an example to allow the reader to grasp the idea.

[1]We joined the **MathML** Working Group of **W3C** in October '99.

Consider the following tautology in propositional logic: $A8744; B8658; (B8658; A)8658; A$. A proof in natural language is "Consider $A8744; B$. If A is true the proof is finished. Otherwise B is true; hence, using the hypotheses $B8658; A$, we can conclude A. QED"

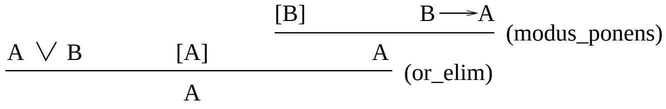

Figure 1: Example of natural deduction tree

Accordingly to Formal Logic, we can think of any proof as a well-formed tree; for example, the previous proof is described in natural deduction style in figure 1. Every node of the tree corresponds to a logical rule, whose name is shown on the right. The root of the tree is the conclusion of the theorem; the un-discharged leaves are the hypothesis. A discharged leaf is a leaf between square brackets; every rule application has associated to it a list of the hypothesis that it discharges. The theorem is correct if every node is well-formed, i.e. it has the right number of children and every child is a proof of the expected proposition with respect to the logical rule. To define a logic, it is enough to specify the set of all the logical rules.

Because working with trees and discharging of the hypothesis is rather cumbersome, logicians have found the way of sequentialising them as expressions. The idea consists in developing a typed 955;-calculus (the core of every functional language) in which every well formed type corresponds to a statement and every well-typed term to a proof. Each logical rule becomes an operator of the calculus of the appropriate type; binding naturally replaces discharging and proof-checking amounts to type-check the corresponding expression.

The above proof, for example, could be sequentialised as $(or_e limH1955; x : A.x955; y : B.(modus_p onensyH2))$

where $H1$ is the hypothesis of type A, $H2$ is the hypothesis of type $B8658; A$, $or_e lim$ has type $A8744; B8658; (A8658; C)8658; (B8658; C)8658; C$ for each A,B and C and $modus_p onens$ has type $A8658; (A8658; B)8658; B$ for each A and B; $955; x : T.M$ is the usual 955;-notation for the function $f(x : T) = M$.

Obviously, once we have the term corresponding to a proof, it is a trivial task to encode its abstract syntax tree into **XML**.

3 XML

The potential relevance of **XML** for the development of large, distributed repositories of formal mathematical knowledge has been already discussed in the introduction. In order to test the actual feasibility of the idea, our first step has been to export in **XML** the *standardlibrary* of a widely used and well-known proof assistant: Coq [7]. Exporting the library was not as easy as one could naively expect. Not only you must wisely choose *what* information is worth exporting, but you must also fight against the several internal intricacies of the application; in particular, some relevant information is not directly available, requiring a tight integration with the system[2]. On the other side, once you have exported all relevant information, you may just forget the underlying application, merely working on **XML**-documents.

[2]A typical problem is the translation from internal names to **URI** (Uniform Resource Identifier), that typically require extra path-information not directly encoded inside the terms; another example, in logical environments encoding proofs with 955;-terms according to the Curry-Howard analogy, is the absence of type information for the inner nodes of the terms, which is essential for rendering purposes.

The standard library of Coq generated about 5200 **XML** files, taking about 120 Mb of disk space (24 Mb after compression). Each file is rather particular, all the information being encoded in markup elements, without any PCDATA content; this is a consequence of the really fine-grained formal encoding, reflected into the DTD. Moreover, the **DOM** (Document Object Model) trees have typically a small branching factor, but a conspicuous depth: it is not unusual to have nesting degrees of several hundred units.

Below is an example of a small fragment of one of this documents, representing the mathematical expression $(n > 0)8744; (0 = n)$.

```
<APPLY>
 <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Disjunction/or.ind"/>
 <APPLY>
  <CONST uri="cic:/coq/INIT/Peano/gt.con"/>
  <REL binder="n" value="1"/>
  <MUTCONSTRUCT noconstr="1" notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
 </APPLY>
 <APPLY>
  <MUTIND notype="0" uri="cic:/coq/INIT/Logic/Equality/eq.ind"/>
  <MUTIND notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
  <MUTCONSTRUCT noconstr="1" notype="0" uri="cic:/coq/INIT/Datatypes/nat.ind"/>
  <REL binder="n" value="1"/>
 </APPLY>
</APPLY>
```

The reading of the previous fragment of code should be quite evident, at least for people acquainted with Coq. The expression is the application of 8221;or" to a pair of arguments. 8221;or" is the first (and, in this case, unique) type defined in the mutual inductive definition contained in the file whose **URI** is `cic:/coq/INIT/Logic/Disjunction/or.ind`. The first argument is an application of the constant 8221;gt" (defined in `cic:/coq/INIT/Peano/gt.con`) to n and to 0 (0 is the first constructor of the datatype of natural numbers, defined in `cic:/coq/INIT/Datatypes/nat.ind`). The second argument is the application of 8221;eq" (Leibniz' equality, defined in `cic:/coq/INIT/Logic/Equality/eq.ind`) to the type of natural numbers, 0 and n.

Note that the information encoded in markup exactly reflects *all* the internal information of Coq; in particular, it reflects all the relevant information, and not just the information required for rendering purposes. In order to prove this assertion, we entirely re-wrote the Coq type-checker to make it work on **XML**-documents. Once the information is extracted and exported in **XML**, it can be used as a more convenient format for exchanging pieces of the library on the Web.

In the sequel, we shall just point out the relevance of our methodology from the point of view of Web publishing, and the re-mathematization of formal content. To this aim, we must first introduce **MathML**.

4 MathML

MathML [11] is an instance of **XML** for describing mathematical expressions capturing both their intended logical content and formatting. The following short introduction largely borrows from the Recommendation, version 2.0.

The **MathML** markup elements fall roughly into two categories: presentation elements and content elements.

4.1 Presentation Elements

The presentation markup of **MathML** provides a quite typical editing environment for mathematical expressions, with no special or distinctive features. The main interest of the language is that it has been recommended by **W3C** as a standard for rendering mathematics on the web and it is likely to be adopted and supported by most browsers. **MathML** presentation markup consists of about 30 elements which accept over 50 attributes. Most of the elements correspond to layout schemata, which contain other presentation elements. Each layout schema corresponds to a two-dimensional notational device, such as a superscript or subscript, fraction or table. In addition, there are the presentation token elements `mi`, `mn` and `mo` that respectively stands for identifiers, numerical constants and operators, as well as several other less commonly used token elements. The remaining few presentation elements are empty elements, and are used mostly in connection with alignment.

The layout schemata fall into several classes. One group of elements is concerned with scripts, and contains elements such as `msub`, `munder`, and `multiscripts`. Another group focuses on more general layout and includes `mrow` (used for grouping subexpressions), `mstyle`, and `mfrac`. A third group deals with tables.

For instance, the expression $(n > 0)8744; (0 = n)$ would be written as follows, whose reading should be clear:

```
<mrow>
  <mrow>
     <mi>n</mi>
     <mo>></mo>
     <mi>0<mi>
  </m:mrow>
  <mo>
    <mchar name="vee"/>
  <mo>
  <mrow>
     <mi>0<mi>
     <mo>=<mo>
     <mi>n</mi>
  </mrow>
</mrow>
```

Note that presentation markup (as well as content markup) is not intended to be edited by hand, but by automatic means or suitable editing tools.

4.2 Content Elements

Content markup consists of about 100 elements accepting a dozen attributes. The majority of these elements are empty elements corresponding to a wide variety of common mathematical operators, relations and functions, such as `partialdiff` (partial differentiation), `leq` (less or equal) and `tan` (tangent)[3]. Others elements, such as `matrix` and `set`, are used to encode various mathematical data types, and a third, important category of content elements such as `apply` are used to build mathematical expressions and also to construct new mathematical objects from others.

[3] **MathML** claims that the base set of content elements should be adequate for simple coding of most of the formulas used from kindergarten to the end of high school and the first two years of college, that is up to A-Level or Baccalaureate level in Europe. Subject areas covered to some extent in **MathML** are: arithmetic, algebra, logic and relations, calculus and vector calculus, set theory, sequences and series, elementary classical functions, statistics, linear algebra.

The apply element is the most important content element, since it provides the only mechanism to build, by composition, complex mathematical expressions. The first child of apply must be an empty element denoting the operator to be applied, with the other child elements as arguments. The operator must be in the range of pre-defined empty elements of **MathML**, or it can be a new user-defined operator, called a csymbol. For instance, the expression $(n > 0)8744; (0 = n)$ is encoded as follows:

```
<apply>
  <or/>
  <apply>
    <gt/>
    <ci>n</ci>
    <ci>0</ci>
  </apply>
  <apply>
    <eq/>
    <ci>n</ci>
    <ci>0</ci>
  </apply>
</apply>
```

Note, by the way, the similarity between this representation and that in Section 3. The similarity can be enforced setting the definitionURL attribute of a **MathML** content element to override the **MathML** default semantics of the element with its formal one in a specific logical system. In the example above we can set the definitionURL attribute of the or operator to the same value (cic:/coq/INIT/LOGIC/Disjunction/or.ind) of the uri attribute of the corresponding low level logical representation.

Each node in the low level, formal representation could, in the general case, be mapped to the root of a subtree in the **MathML** representation. Hence, in order to use **MathML** as an intermediate language, it is also necessary to preserve a link from the root of each subtree to the possibly corresponding low level element; in this way, any interaction with the **MathML** markup could be reflected onto the formal document.

4.3 Mixing Content and Presentation

The third category of elements is meant to supply mechanisms for mixing presentation and content markup by embedding one into the other, or by establishing bindings via so called 8221;semantic mappings". One common use for the semantics element is to bind a piece of content markup to some presentation markup as a semantic annotation. In this way, an author can specify a non-standard notation to be used when displaying a particular content expression.

The **MathML** specification insists to keep both content and presentation markup into a single document. Although it is clear that both aspects are tightly related in mathematics, there is no major evidence why this relation should eventually traduce into a physical dependency inside a single document. In one direction, we can imagine to define the relation from content to presentation by means of stylesheets (see next section), and conversely preserve the relation from presentation to content by means of **XLink**s [13], [14], [15], that is a solution (almost) compatible with the current specification.

5 XSL Transformations

As already mentioned in the introduction, we advocate the use of stylesheets as a standard mechanism associating notation to content for mathematical documents. The choice of **XSLT** is naturally motivated by the fact that **XSLT** has been specifically designed to generate the styling of **XML** documents. Moreover, a strong proviso of our project is the exploitation of only standard **XML** technologies to improve re-usability and integration of software components.

However, the impact of this choice from the points of view of complexity, code readability and maintainability was not evident a priori and it actually revealed quite cumbersome (we believe we have really pushed **XSLT** to its extreme possibilities). We will discuss this issues in section 7.1

At the time we started our project, there existed already some efforts to apply **XSLT** to generate **MathML** presentation markup from **MathML** content markup. We have taken advantage of these works, embedding a stylesheet defined by Igor Rodionov, of the Computer Science Department of the University of Western Ontario, London, Canada. Here is a fragment of it (the code has been slightly simplified, for the sake of clarity).

```
<xsl:template match = "m:apply[m:or[1]]">
  <xsl:param name="IN_PREC" select="$NO_PREC"/>
  <xsl:choose>
    <xsl:when test="$IN_PREC > $OR_PREC">
      <m:mfenced separators="">
        <xsl:apply-templates select="." mode="or"/>
      </m:mfenced>
    </xsl:when>
    <xsl:otherwise>
      <m:mrow>
        <xsl:apply-templates select="." mode="or"/>
      </m:mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match = "m:apply[m:or[1]]" mode="or">
  <xsl:apply-templates select="*[2]" mode = "semantics">
    <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
  </xsl:apply-templates>
  <xsl:for-each select = "*[position()>2]">
    <m:mo> <m:mchar name="vee"/> </m:mo>
    <xsl:apply-templates select="." mode = "semantics">
      <xsl:with-param name="IN_PREC" select="$OR_PREC"/>
    </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>
```

When extending **MathML** content by means of user-defined `csymbols`, we have just to extend the previous stylesheet to cover the new particular cases (and **XSLT** has a clean inclusion mechanism to deal with this kind of extensions).

The transformation from **MathML**-content to **MathML**-presentation requires less than 2000 lines of **XSLT**. The extension to new, Coq-oriented `csymbols`, requires 1400 additional lines. We also defined a bunch of stylesheets generating **HTML** from **MathML**-content (about 1500 lines).

5.1 From the low-level logical specification to MathML content

Of course, we can use stylesheets to pass from the low-level formal representation of the information (in a specific logical dialect) to **MathML** content. By composition, this also implicitly defines an intended presentation for it.

When translating a specific logical dialect (like that of Coq) into **MathML** content, it is very likely that almost any specific construct of the dialect will become a new `csymbol`. Typically, **MathML** content starts to show its potentiality in the encoding of *defined* entities.

For instance, the **gt** (greater than) relation is not a primitive notion in Coq: actually, it is a suitable inductive definition. However, when we pass from Coq to **MathML** there is no point to preserve this information, if not as a pointer to its actual definition. So, the application of Coq-**gt** constant to its pair of arguments can be directly transformed into an application of **MathML-gt** element to (the result of the transformation on) its arguments, automatically recovering the intended presentation.

The following templates captures this idea:

```
<xsl:template match="APPLY[CONST[attribute::uri='cic:/coq/INIT/Peano/gt.con']
                            and (count(child::*) = 3)]" >
   <m:apply>
   <m:gt definitionURL="{CONST/@uri}"/>
    <xsl:apply-templates select="*[2]"/>
    <xsl:apply-templates select="*[3]"/>
   </m:apply>
</xsl:template>
```

In a similar way, content (and notation) can be associated to all user-defined notions. In case the notion is not in the primitive set of **MathML**-content, it is enough to create a new `csymbol`, and extend the presentation stylesheet with the intended presentation.

The general idea is that a bunch of 8221;basic" stylesheets should be available to cover standard parts of the library, while each new contributor should also be responsible to define its own stylesheet to fix its own personal notation (or to override an existent one). This is made possible by the machinery of XSLT: `import` and `include` priority policies allow to specify different notational precedences by combining user defined stylesheets. Unfortunately, this mechanism is not dynamic, forcing a re-compilation of the stylesheets every time the tree of combined stylesheets changes.

The development of stylesheets from the low-level **XML** representation into **MathML**-content is still under developments. At present we just cover the main logical connectives, and the basic notation for number theory, set-theory and reals (for a total of about 1500 lines of **XSLT**).

6 Beyond expressions

MathML is only concerned with mathematical expressions. So, we must also face the problem to deal with different mathematical entities, like proofs, definitions, lemmas, theorems, hypotheses and their scope.

Proofs are peculiar. On one side we can consider them as a particular category of mathematical expressions (this is the essence of proof theory), and they can be actually encoded into **MathML**-content with no much effort. On the other side, simple notational or pretty-printing mechanisms do not help that much in improving the reading of complex formal derivations. Two ways seem to be possible, here:

- define complex stylesheets attempting some kind of automatic translation of formal proofs into natural language, along the lines of systems like the *Natural* module of Coq [5], [4], [7]. Again, for obvious portability reasons, it would be much more interesting to have these transformations expressed as a stylesheet than as a Coq module.
- associate explicit annotations in natural language to the low level structured representation of the information. In this case, the stylesheet is merely responsible to compose the various pieces of annotation into a single statement.

We are pursuing both approaches in parallel. We just spend a few words on the first, here, since it is more closely related to the topic of this paper.

The general idea is that we should heavily work on the structure of the proof, before attempting its translation. There are two main operations to be done:

1. recognise the application of usual logical principles (not primitive in the system!), such as an introduction or an elimination of a conjunction, say, or the use of an induction schema. All these logical constructions typically require a specific presentation (and content description).
2. Change the structure of the proof, looking for 8221;subproofs" which must be brought to the surface: a 955;-term encodes a 8221;bottom-up" description of the proof, from the conclusion to the premises; however, we would expect to have a 8221;top-down" presentation, from the premises to the goal.

An important and, apparently, original notion that is emerging from our work on this topic is that of 8221;logical thread". A logical thread is a sequence of 8221;linear" logical steps where each one just depends from the result of the previous one. Similarly, we must be able to distinguish between 8221;major" and 8221;minor" premises. For instance consider the application of a rewriting rule transforming a proof of $P(t1)$ in a proof of $P(t2)$. This just depends from the proof of $P(t1)$ (major premise) and a proof of $t1 = t2$ (minor premise, typically presented aside). In our terminology, we say that rewriting is a pseudo-linear logical operation, that is a logical rule with a single major premise and one (or more) minor premises.

In figure 2 you may find an example of a simple proof borrowed from the standard library of Coq after the application of these proof transformations (and notational stylesheets). The Postscript was *automaticallygenerated* from **MathML** presentation, using one of the features of our GtkMathView widget (`http://www.cs.unibo.it/helm/mml-widget`). Note that, at **MathML** level, most of the symbols in Figure 2, and all constant names are hyperlinks to the corresponding definitions (so, using a browser aware of **XLink**s - such as the Gtk-widget - you may directly jump from an occurrence of a name/symbol to its actual definition).

The textual part is deliberately kept to a bare minimum, especially in this prototyping phase. For instance it would be very simple to change a 955; $x : T$ into a more appealing sentence in natural language such as 8221;assume x of type T", or 8221;let x be a generic object of type T", but these are marginal details. Of course, many other improvements can still be done, but the example should already give a gist of the power of our techniques.

As a comparative example, we also include the Coq term encoding this proof.

Coq term

```
Rgt_Ropp =
[r1,r2:R; H:(Rgt r1 r2)]
```

DEFINITION Rgt_Ropp() OF TYPE

$\forall r_1 : R.\ \forall r_2 : R.(r_1 > r_2 \to -r_1 < -r_2)$

AS

$\lambda r_1 : R$

$. \lambda r_2 : R$

$. \lambda H : r_1 > r_2$

$.(h_1)$(Rplus_ne $-r_1$)

we proved $-r_1 + 0 = -r_1 \wedge 0 - r_1 = -r_1$

In particular, we have

$(a)\ -r_1 + 0 = -r_1$

$(b)\ 0 - r_1 = -r_1$

(Rplus_ne $-r_2$)

we proved $-r_2 + 0 = -r_2 \wedge 0 - r_2 = -r_2$

In particular, we have

$(a_0)\ -r_2 + 0 = -r_2$

$(b_0)\ 0 - r_2 = -r_2$

$\lambda H_1 : -r_1 < -r_2 . H_1$

we proved $\forall H_1 : -r_1 < -r_2 . -r_1 < -r_2$

Rewrite $-r_2$ with $0 - r_2$ by b_0

we get $(-r_1 < 0 - r_2 \to -r_1 < -r_2)$

Rewrite $-r_1$ with $-r_1 + 0$ by a

we get $(-r_1 + 0 < 0 - r_2 \to -r_1 < -r_2)$

Rewrite 0 with $-r_1 + r_1$ by (Rplus_Ropp_l r_1)

we get $(-r_1 + 0 < -r_1 + r_1 - r_2 \to -r_1 < -r_2)$

Rewrite $-r_1 + r_1 - r_2$ with $-r_1 + r_1 - r_2$ by (Rplus_assoc $-r_1$ r_1 $-r_2$)

we get $(-r_1 + 0 < -r_1 + r_1 - r_2 \to -r_1 < -r_2)$

Rewrite $r_1 - r_2$ with $-r_2 + r_1$ by (Rplus_sym $-r_2$ r_1)

we get $(-r_1 + 0 < -r_1 + -r_2 + r_1 \to -r_1 < -r_2)$

(h_2)(Rplus_Ropp_l r_2)

we proved $-r_2 + r_2 = 0$

(h_3)(Rlt_compatibility $-r_2$ r_2 r_1 H)

we proved $-r_2 + r_2 < -r_2 + r_1$

(Rlt_compatibility $-r_1$ $-r_2 + r_2$ $-r_2 + r_1$ previous)

we get $-r_1 + -r_2 + r_2 < -r_1 + -r_2 + r_1$

Rewrite 0 with $-r_2 + r_2$ by h_2 in h_1 and apply to h_3

we proved $-r_1 < -r_2$

we proved $\forall r_1 : R.\ \forall r_2 : R.\ \forall H : r_1 > r_2 . -r_1 < -r_2$

Figure 2: Rendering of Coq term

```
(eqT_ind_r R R0
  [r:R]
   (Rlt (Rplus (Ropp r1) r) (Rplus (Ropp r1) (Rplus (Ropp r2) r1)))
   ->(Rlt (Ropp r1) (Ropp r2))
  (eqT_ind_r R (Rplus r1 (Ropp r2))
    [r:R]
     (Rlt (Rplus (Ropp r1) R0) (Rplus (Ropp r1) r))
     ->(Rlt (Ropp r1) (Ropp r2))
    (eqT_ind R (Rplus (Rplus (Ropp r1) r1) (Ropp r2))
      [r:R](Rlt (Rplus (Ropp r1) R0) r)->(Rlt (Ropp r1) (Ropp r2))
      (eqT_ind_r R R0
        [r:R]
         (Rlt (Rplus (Ropp r1) R0) (Rplus r (Ropp r2)))
         ->(Rlt (Ropp r1) (Ropp r2))
        (and_ind (Rplus (Ropp r1) R0)==(Ropp r1)
          (Rplus R0 (Ropp r1))==(Ropp r1)
          (Rlt (Rplus (Ropp r1) R0) (Rplus R0 (Ropp r2)))
          ->(Rlt (Ropp r1) (Ropp r2))
          [a:((Rplus (Ropp r1) R0)==(Ropp r1));
           _:((Rplus R0 (Ropp r1))==(Ropp r1))]
           (eqT_ind_r R (Ropp r1)
```

```
              [r:R]
               (Rlt r (Rplus R0 (Ropp r2)))->(Rlt (Ropp r1) (Ropp r2))
              (and_ind (Rplus (Ropp r2) R0)==(Ropp r2)
                (Rplus R0 (Ropp r2))==(Ropp r2)
                (Rlt (Ropp r1) (Rplus R0 (Ropp r2)))
                ->(Rlt (Ropp r1) (Ropp r2))
                [_:((Rplus (Ropp r2) R0)==(Ropp r2));
                 b0:((Rplus R0 (Ropp r2))==(Ropp r2))]
                 (eqT_ind_r R (Ropp r2)
                   [r:R](Rlt (Ropp r1) r)->(Rlt (Ropp r1) (Ropp r2))
                   [H1:(Rlt (Ropp r1) (Ropp r2))]H1
                   (Rplus R0 (Ropp r2)) b0) (Rplus_ne (Ropp r2)))
              (Rplus (Ropp r1) R0) a) (Rplus_ne (Ropp r1)))
        (Rplus (Ropp r1) r1) (Rplus_Ropp_l r1))
      (Rplus (Ropp r1) (Rplus r1 (Ropp r2)))
      (Rplus_assoc (Ropp r1) r1 (Ropp r2))) (Rplus (Ropp r2) r1)
    (Rplus_sym (Ropp r2) r1)) (Rplus (Ropp r2) r2) (Rplus_Ropp_l r2)
  (Rlt_compatibility (Ropp r1) (Rplus (Ropp r2) r2)
    (Rplus (Ropp r2) r1) (Rlt_compatibility (Ropp r2) r2 r1 H)))
    : (r1,r2:R)(Rgt r1 r2)->(Rlt (Ropp r1) (Ropp r2))
```

Finally, let us remark that we are not responsible for the proof, that could be easily improved. As a matter of fact, the emphasis devoted in recent times to tactics (provability), and the substantial impossibility to read the proof-object resulting from a sequence of tactics have naturally lead to generate huge libraries of 8221;bad" proofs (you have much worse examples than the one above, in Coq standard library). Our tool, providing a way to read proof objects in a much more natural way, should help both in improving the quality of proofs, and in improving the actual implementation of tactics.

6.1 The Document Structure

Definitions, theorems and other mathematical notions eventually require a further level, distinct from **MathML** (let us call it the *Document* level). The interesting point is that this level (as well as a further level of metadata), can probably be entirely standardised, being largely independent from the specific foundational dialect. Our work in this direction is still preliminary; an interesting proposal addressing these issues is provided by **OMDoc** (Open Math Document) [12].

7 The Global Picture

Figure 3 summarises the global architecture of the transformation process described in the previous sections.

Let us stress again that passing through an intermediate representation like **MathML** content is particularly important for the modularity of the overall architecture: many specific logical dialects can be mapped into the same intermediate language (or into suitable extensions of it), and many transformations to specific editing formats can be defined for the single intermediate language. Just simple extensions of the transformations would be required, in case of extensions of the intermediate representation.

7.1 Problems and solutions analysis

As a whole, all the developed stylesheets amount to about 9000 lines of **XSLT**. Rendering times are now acceptable, in the order of a few seconds for medium-sized

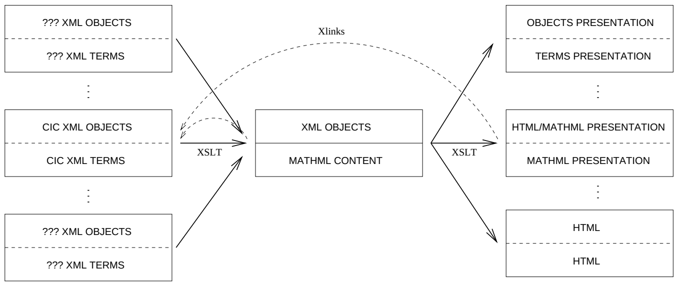

Figure 3: The global picture

input files. Achieving these performances has required many deep re-designs of the stylesheet architecture and accurate optimisations of the code (the original mean rendering time was in the order of some minutes!). The most influents have been:

1. **The introduction of unique identifiers for each node in the source documents.** **XPath** expressions to identify a node have in the general case a length linear in the node depth; hence, due to the conspicuous nesting depth of the source **DOM** trees, their computational and spatial complexity were unmanageable. For this reason, we have added unique identifiers to each node of the source **XML** documents to simplify **XPath** expressions, that are all trivial.

2. **Using variables to hold external document fragments.** A quite astonishing performance improvement derived by splitting input documents, applying the stylesheet only on the main fragment and loading the others as node-sets into **XSLT** variables.

   This is only one representative of a large set of little modifications on the stylesheets that lead to unexpected performance changes. Another example is the usage of **XSLT** key: for our set of stylesheets, substituting keys with an explicit linear search on a flat tree unexpectedly was more performing than **Xalan** key implementation.

   The above ones seem to be idiosyncrasies of the implementation of the particular **XSLT** engine we are using; generally speaking, though, the great sensibility of the performance to even little changes in the stylesheets is rather annoying.

Other critical issues are stylesheets readability and maintainability. To keep the complexity manageable and improve scalability, we are successfully trying the following solutions:

1. **Replacing of the single application of a complex stylesheet with several simple stylesheet applications.** The principal source of complexity for stylesheets is the existence of multiple templates with the same matching pattern to perform different tasks; at run time, the activated template is chosen depending on the specified `XSLT` mode and precedences. So, understanding the run-time behaviour of a stylesheet become difficult.

   To avoid the previous situation and reduce the high number of different modes or precedence levels, we can split the single stylesheet into as many ones as the number of tasks to perform. Each one is applied on the result of the previous one.

A different usage is to encompass the impossibility to process in a same run the output generated (the well-known result tree fragment limitation). To achieve this result, we need an **XSLT** engine providing an **HTTP** interface, as our UWOBO stylesheet manager (see next section). By means of it, we can require, via the `document` function, the application of a stylesheet to a document, in order to process its output.

2. **Automatic generation of notational stylesheets** The main user contributions expected are the notational stylesheets; hence, the fact of providing them in a way as simple as possible is fundamental. Luckily, notational stylesheets have a repetitive and quite uniform structure: for example, all the templates to associate a particular notation to a binary operation are almost identical. So, we are going to provide a stylesheet to automatically generate notational stylesheets starting from a trivial user-defined **XML** description of the notations, mainly comprising the **URI**s of the operators, their arities and the corresponding **MathML** content element.

   We have still to understand which amount of user provided stylesheets could be automatically generated. In particular, this seems much more difficult for the templates overriding the default natural language rendering.

We still suffer from well-known limitations of **XSLT** and **XPath**, such as the limited support to regular expression functionalities for text processing, the differences between node-sets and result tree fragments, the impossibility to define keys on documents stored as node-sets, the difficulty in using keys on documents different from the current one.

8 Current State

The library is accessible [4] at the **URL** (Uniform Resource Locator) `http://www.cs.unibo.it/helm/library.html`. Each **XML** file of Coq standard library can be currently consulted in several different ways, corresponding to different stylesheets applications. Stylesheets are applied *onthefly* to source **XML** documents; stylesheet parameters can be passed at application time to fine tune the requested rendering.

All the processing is done on the server side using UWOBO (`http://www.cs.unibo.it/helm/uwobo`). UWOBO is a Xalan[5] based servlet which is capable of applying several subsequent stylesheets to **XML** documents located anywhere over the net. The source document, the stylesheets to apply and the stylesheets parameters are specified in the dynamic part of the **URL** used to contact the processor.

In a first prototype implementation we used Cocoon[6] for the processing of on-line documents. This solution had several disadvantages, notably the lack of flexibility when the same source **XML** document is going to be applied to different stylesheet sequences depending on the user choices, as in our cases. Moreover, Cocoon has been designed to be a complete server-side publishing framework, whereas UWOBO addresses only the task of stylesheet application and management; thus it is small enough to make feasible its use also in client-side processing so to reduce network communications.

User interaction with the library is allowed via client-side **DHTML** (Dynamic HTML) only, implemented by means of a suitable combination of JavaScript events and invocations to the UWOBO processor. The main role of JavaScript consists

[4] Please, be aware that the system is evolving daily, and you could easily meet problems due to temporarily broken files or configurations.
[5] `http://xml.apache.org/xalan`
[6] `http://xml.apache.org/cocoon/index.html`

in generating on-the-fly the **URL**s to control UWOBO; such **URL**s depend on the user choices, as, for example, the rendering format required. Currently, the following output formats are provided:

1. **The source XML file.** It is possible to download the file both in compressed and uncompressed format, independently from its format on the server. In fact, the verbosity of **XML** induced us to use also a compressed format for the library documents. The un-compression is possibly done on-the-fly at each request.

2. **The MathML content format.**

3. **The MathML presentation format.** Our **MathML** documents are some way peculiar, being made of huge formulas representing proofs which usually spans over many table lines; this require a fine-tuned control on the final layout, based on many levels of nested tables. Current browsers claiming to be **MathML** compliant (such as Amaya and Mozilla) still have some problems rendering such complex expressions; so we have developed our own **MathML** browser (as a Gtk-widget) fully compliant to the Recommendation of **MathML** 2.0. The widget is a quite complex application (about 30.000 lines of C++), also supporting quite distinctive forms of interaction.

4. **The HTML format.** Of course, this format provides just an approximation of the expected rendering. When a good sets of fonts is installed[7], though, the result is satisfactory enough and makes the library available to a wide basin of users.

9 Conclusions

This work is part of a larger project aimed to exploit the potentiality of **XML** technology for the creation and maintenance of an electronic, distributed, hypertextual library of formal mathematical knowledge. More details on the project can be found at the **URL** `http://www.cs.unibo.it/helm`. This paper is mainly focused on (Web-)publishing issues, stressing the potentiality offered by a joint use of Stylesheets and **MathML**. In particular, **MathML** provides both a powerful presentational language supporting complex two-dimensional mathematical notation, and an interesting 8221;content" level that can be profitably used, with a relevant improvement in modularity, as an intermediate representation of the information.

From the point of view of transformations, **XSLT** provides a standard, extensible and application independent language for expressing notation. In this perspective, our role is the development of a suitable architecture for the processing of mathematical documents by means of stylesheets. Moreover, we are developing a modular core library of stylesheets for the default rendering of documents; the aim of this library is to hide the stylesheet complexity, allowing anyone to add new notations and change the default rendering in a simple way. We believe that the development of these specialised notations could (and should) be conceived as a joint effort of the whole proof assistant community.

References

[1] A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. $Content - centricLogicalEnvironments$. Short Communication at the Fifteenth

[7]Unix/Linux Netscape users must enable the right font-set, adding the following line to their `.Xdefaults` file: `Netscape*documentFonts.charset*adobe-fontspecific: iso-8859-1`

Annual IEEE Symposium on Logic in Computer Science (LICS'2000), June 26 - 29, 2000, Santa Barbara, California.

[2] A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. $Formal Mathematics in MathML$. Proceedings of the first International Conference on MathML and Math on the Web (MathML 2000). October 20-21, 2000, Urbana-Champaign, IL, USA.

[3] A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. $Towards a library of formal mathematics$. Panel session of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLS'2000), Portland, Oregon, USA.

[4] Y. Coscoy. $Explication textuelle de preuves pour le calcul des constructions inductives$. PhD Thesis, Universit de Nice - Sophia Antipolis, 2000

[5] Y.Coscoy, G.Kahn, L.Thery. $Extracting Text from Proofs$. Technical Report RR-2459, INRIA Sophia Antipolis, 1995.

[6] Girard, Lafont, Taylor. $Proofs and Types$ Cambridge Univ. Press, Cambridge Tracts in Th. Comp. Sc., 1989.

[7] The Coq Proof Assistant Reference Manual. Version 6.3. INRIA Internal Report. 1999.

[8] eXtensible Markup Language (XML). Version 1.0, W3C Recommendation February 1998. **http://www.w3.org/XML**.

[9] Document Object Model (DOM) Level 2 Specification. Version 1.0, W3C Candidate Recommendation 13 November, 2000. **http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/**

[10] XSL Transformations (XSLT). Version 1.0, W3C Recommendation 16 November 1999. **http://www.w3.org/TR/xslt**.

[11] Mathematical Markup Language (MathML) 2.0 W3C Recommendation, 21 February 2001. **http://www.w3.org/TR/MathML2/**.

[12] Open Mathematical Documents (OMDoc) 1.0, November 1 2000. **http://www.mathweb.org/omdoc**.

[13] XML Linking Language (XLink) Version 1.0. W3C Candidate Recommendation 3 July 2000.

[14] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999. **http://www.w3.org/TR/xpath**

[15] XML Pointer Language (XPointer) Version 1.0. W3C Candidate Recommendation.