HELM and the Semantic Math-Web

Andrea Asperti, Luca Padovani, Claudio Sacerdoti Coen, Irene Schena
Department of Computer Science
Via di Mura Anteo Zamboni 7, 40127 Bologna, ITALY.
contact: `asperti@cs.unibo.it`

Abstract

The eXtensible Markup Language (XML) opens the possibility to start anew, on a solid technological ground, the ambitious goal of developing a suitable technology for the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge. In particular, XML provides a central technology for storing, retrieving and processing mathematical documents, comprising sophisticated web-publishing mechanisms (stylesheets) covering notational and stylistic issues. By the application of XML technology to the large repositories of structured, content oriented information offered by Logical Frameworks we meet the ultimate goal of the Semantic Web, that is to allow machines the sharing and exploitation of knowledge in the Web way, i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

1 Introduction

Existing logical systems are not suitable for the creation of large repositories of structured mathematical knowledge accessible via Web. In fact, libraries in logical frameworks are usually saved in two formats: a textual one, in the specific tactical language of the proof assistant, and a compiled (proof checked) one in some internal, concrete representation language. Both representations are clearly unsatisfactory, since they are too oriented to the specific application: the information is not directly available, if not by means of the functionalities offered by the system itself. This is in clear contrast with the main guidelines of the modern Information Society, the recent emphasis on *content* and the new frontier of the so called "Semantic Web[1].". The goal of the Semantic Web is to pass from a "machine readable" to a "machine understandable" encoding of the information: establishing a layer of machine understandable data would allow automated agents, sophisticated search engines and interoperable services and will enable higher degree of automation and more intelligent applications.

In contrast with current digital encodings of mathematical information (e.g. in digital libraries), which are "machine-readable" but not "machine-understandable", Logical Frameworks offer *huge* repositories of structured, content oriented information, naturally providing a major arena for the Semantic Web and its technologies. The point is to allow access to this mathematical knowledge in the "Web way", i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

The first, mandatory step in this direction is the direct encoding of the libraries in XML, which has recently imposed as the main tool for representation, manipulation, linking and exchange of structured information. But of course the broad goal of the Semantic Web goes far beyond the trivial suggestion to adopt XML as a neutral specification language for

[1] Semantic Web Activity, `http://www.w3.org/2001/sw`

the "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. Here is a short list of the added-value offered by the XML approach:

Standardization 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 searching, retrieving, displaying or authoring (just to mention a few of them) that are largely independent from the specific logical system.

Publishing XML offers sophisticated web-publishing technologies (Stylesheets, MathML, ...) which can be profitably used to solve, in a *standard* way, the annoying notational problems that traditionally afflict formal mathematics.

Searching & Retrieving The World Wide Web is currently doing a big effort in the Metadata and Semantic Web area. Languages as the Resource Description Framework or XML-Query are likely to produce innovative technological solutions in this field.

Interoperability 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.

Modularity The "xml-ization" process should naturally lead to a substantial simplification and re-organization of the current, "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 *content-based* architecture of future systems.

In this article we present our project on the use of XML technology for the development and maintenance of distributed repositories of formal mathematical knowledge: the Hypertextual Electronic Library of Mathematics (HELM[2]).

The HELM project has already been presented as a "work in progress" in the last edition of TPHOLs [1]: this paper is a revisitation of the previous work, containing both assessed results and new achievements of the project during the last year.

2 The eXtensible Markup Language

Perhaps, the best way to introduce XML in few lines is to take a look at a simple example. The following XML document is a possible encoding of the definition of the inductive type of natural numbers in Coq [15].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE InductiveDefinition SYSTEM ""http://www.cs.unibo.it/helm/dtd/cic.dtd">
<InductiveDefinition noParams="0" params="">
 <InductiveType name="nat" inductive="true">
  <arity><SORT value="Set"/></arity>
   <Constructor name="O">
    <REL value="1" binder="nat"/>
   </Constructor>
  <Constructor name="S">
   <PROD>
```

[2] http://www.cs.unibo.it/helm

```
    <source><REL value="1" binder="nat"/>
    </source>
    <target><REL value="2" binder="nat"/>
    </target>
   </PROD>
  </Constructor>
 </InductiveType>
</InductiveDefinition>
```

XML gives a method for structuring data in a text file. Roughly speaking, the XML specification says that a XML document is made of *tags* (words bracketed by '<' and '>'), *attributes* (of the form `name="value"`) and text. Tags are used to delimit *elements*. Elements may appear in one of the following two forms: either they are non-empty elements, as `InductiveDefinition` or `InductiveType` (they can contain other elements or text), or they are empty elements, as `SORT` or `REL`. The previous example contains no text: this is a peculiarity of really *formal* encodings, where every bit of knowledge has a specific intelligence worth to be encapsulated in markup.

The XML specification defines a XML document to be well-formed if it meets some syntactical constraints on the use of tags and attributes. For example, non-empty elements must be perfectly balanced. For this reason, someone can think of tags of non-empty elements as labelled brackets for structuring the document.

It is remarkable that XML does not specify any predefined tag set at all. Rather, it lets the user specify his own grammar by means of a Document Type Definition[3] (DTD), a document which defines the allowed tags, the related attributes and which is the legal content for each element. The XML specification just defines the validity of a XML document with respect to a given DTD. This is why XML is a *meta-language* that can be instantiated to a potentially infinite set of languages, each with its own DTD.

For example, here is a DTD fragment for the previous xml file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<!ENTITY % term '(LAMBDA|CAST|PROD|REL|SORT|APPLY|VAR|META|IMPLICIT|CONST|
                  LETIN|MUTIND|MUTCONSTRUCT|MUTCASE|FIX|COFIX)'>
...
<!ELEMENT InductiveDefinition (InductiveType+)>
<!ATTLIST InductiveDefinition
          noParams NMTOKEN #REQUIRED
          params   CDATA   #REQUIRED
          id       ID      #REQUIRED>
<!ELEMENT InductiveType (arity,Constructor*)>
<!ATTLIST InductiveType
          name      CDATA        #REQUIRED
          inductive (true|false) #REQUIRED>
<!ELEMENT Constructor %term;>
<!ATTLIST Constructor
          name CDATA #REQUIRED>
...
```

A `InductiveDefinition` element may contain one or more `InductiveType` elements, each `InductiveType` contains an `arity` followed by a possibly empty sequence of `Constructors`, and a `Constructor` is an arbitrary `term` (an `ENTITY` is a macro declaration). `noParams`,

[3]DTDs are going to be replaced by XML Schemas which provide more detailed descriptions of the syntactical and semantical constraints for a XML document. Moreover, Schemas are themselves XML documents and can then be manipulated and enriched using the same XML technologies.

`params` and `id` are attributes of `InductiveDefinition`, and `name` is the only attribute of `Constructor`. The keyword `REQUIRED` states that an attribute is mandatory, i.e. it cannot be omitted when using its related tag.

References to Documents Documents and resources in general must have a name in order to be accessible over the Web. This is accomplished via the use of URIs (Universal Resource Identifiers) as defined in [2]. A generic URI is made of a formatted (structured) string of characters whose intended meaning is associated by the applications managing it. URLs (Uniform Resource Locators) are a particular kind of URIs specifically designed to name resources accessed by means of a given standard protocol (for example the HTTP protocol). URLs consist of a first part identifying the protocol and a host followed by a second part to locate the resource on it.

URLs can be resolved by standard processing tools and browsers, but suffer from problems of consistency: moving the target document leads to dangling pointers; moreover, being physical names, they cannot be used to identify a whole set of copies located on different servers for fault-tolerance and load-balancing purposes. URIs, instead, can be designed as logical names, leaving to applications the burden of resolution to physical names. So, for examples, the URI "`cic:/Coq/Reals/Rdefinitions/R.con`" could be used as a logical name for the axiom which defines the existence of the set `R` of real numbers in the standard library of the Coq Proof Assistant; then, an application is required to map the URI to a physical name (an URL) as "`http://coq.inria.fr/library/Reals/Rdefinitions/R.con.xml`".

3 The HELM Project

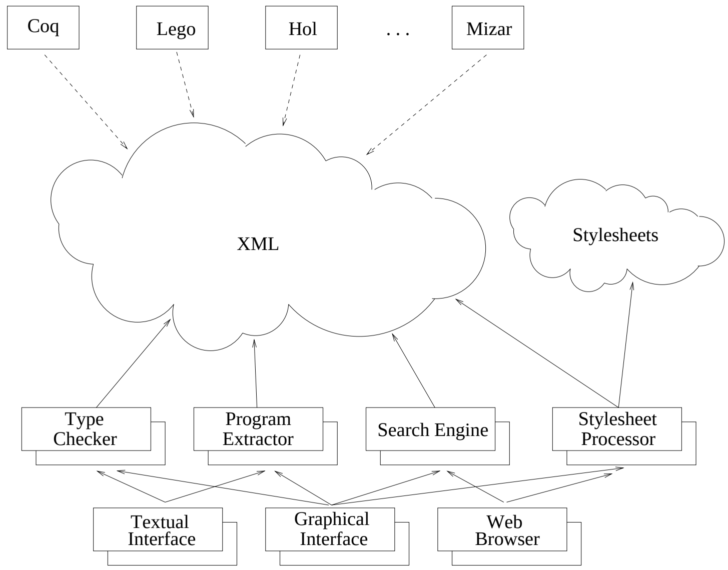

Figure 1: Architecture of the HELM project.

The overall architecture of our project is depicted in Figure 1.

Once XML has been chosen as the standard encoding format[4] for the library, we must face the problem of recovering the already codified mathematical knowledge. Hence, we need new

[4]A standard *format*, not a standard *language*! In other words, the standardization we are pursuing is not

modules implementing exporting functionalities toward the XML representation for all the available tools for proof reasoning. Currently, we have just written such a module only for the Coq proof assistant [15]. In the near future, we expect that similar exporting functionalities will be provided by the developers of the other logical systems. We will describe the exporting issues in section 4.

To exploit and augment the library, we need several tools to provide all the functionalities given by the current "monolithic" proof assistants, such as type-checking, proof searching, program verification, code extraction and so on. Moreover, we can use the available well-developed and extensible tools for processing, retrieval and rendering of XML-encoded information. In particular, to render the library information, we advocate the use of stylesheets, which are a standard way of associating a presentation to the content of XML files. This allows the user to introduce new mathematical notations by simply writing a new stylesheet. In section 5 we shall briefly discuss our implementation of a type-checking tool, while in section 6 stylesheets are addressed in details.

The user will interact with the library through several interfaces that integrate the different tools to provide an homogeneous view of the functionalities. We are developing two interfaces, described in section 7.

Because of the particular nature of the library, we have also provided a suitable model of distribution, which is discussed in more detail in section 8.

4 Exporting from Coq

Coq is one of the most advanced proof-assistants, based on a very rich logical framework called the Calculus of (Co)Inductive Constructions (CIC). The great number of functionalities (proof editing, proof checking, proof searching, proof extraction, program verification) have made the system, whose last stable release is V6.3.1, very big and complex. In fact, the only practical way to work with the information encoded in Coq is that of writing a new module that extends the system gaining access to its internal representation.

A new release of Coq, called V7, is being now developed with the precise aim to reduce the implementation complexity; notwithstanding this, finding the right information inside the system itself is not a trivial task: first, information is encoded in Coq's data structures which change among different versions of the system; second, the information searched is often not directly available. For example, when a file is loaded in V6.3.1, its path is completely forgotten, while we would like to export files in directories reflecting the logical structure of the library. Due to these difficulties, Coq has proven itself a challenging test bench for exporting information to XML.

To do the exporting, we have written a module for V6.3.1 in which we have implemented a set of top-level commands that, given the name of one or more CIC objects (variables, constants, inductive types or axiom definitions), generate a corresponding XML file. Due to the previous problem of missing information, the actual exporting is done in two distinct phases: in the first one the XML files are exported from Coq; in the second phase, implemented outside Coq, they are moved into their proper location and all the URIs are fixed to reflect these changes.

To overcome these difficulties, we are having a collaboration with the Coq team to modify the system to solve all the problems we encountered in V6.3.1. Moreover, starting from the forthcoming version 7.0 of Coq, our exportation module will become a component of the standard distribution. As another consequence of the collaboration, the proof engine of Coq will be improved borrowing from project HELM the idea of using absolute names for the elements of the library that will be no more flat, but structured into theories and sub-theories according to the absolute names.

at the *logical* level, but at the *technological* one.

In the rest of this section we only focus on the implementation for version 6.3.1 because it is more representative of the difficulties that could be faced when exporting from an existing proof-assistant without the collaboration of its development team.

4.1 Relevant Information to be Exported

To design the module, the first difficulty has been the identification of which information should be exported and what should be its structure. We have chosen not to export:

Parsing and pretty-printing rules Parsing rules should depend only on the proof engine. To be able to use other proof engines different from Coq we cannot rely on Coq's own rules. Similarly, pretty-printing rules should depend only on the users choice and the type of available browser.

Tactics-related information These, too, are proof engine dependent. Moreover, we do not think that the tactics used to do a proof are really meaningful to understand the proof itself (surely, they are not the real informative content). In fact, the level of tactics and the level at which a proof should be understood are not the same: what some simple tactics do (as "Auto" that automatically search a proof) is not at all obvious. Moreover, the sequence of tactics used is clearly reflected in the λ-term of the proof; hence it is possible to add as an attribute to a subterm the name and parameters of the tactic used to generate it.

Redundant information added by Coq to the terms of CIC Coq adds in several places a lot of information to CIC terms in order to speed up the type-checking. For example, during the type-checking of an inductive definition, Coq records which are the recursive parameters of its inductive constructors; this information is then used during the type-checking of fix functions to ensure their termination. This is an example of a clearly redundant information, surely useless for browsing purposes or for indexing, but necessary to every proof-checker implementation. Other times, instead, the added information is important only for browsing purposes. For example, sometimes the user asks the system to infer a type and probably does not want to view the inferred type thereafter.

So, we have decided to export a first core of information, roughly corresponding to the only one available at the logical level, using a **principle of minimalism**: *no redundant information should be exported.* If the principle were not followed, in every application loading an XML file we would have to add checks to verify the consistency of the redundant (and possibly useless!) information.

In section 4.2 we will see how this does not prevent us to link in a precise way additional information to the core one.

The remaining, interesting information could be structured into three different levels that we have clearly separated in the XML representation. The first one is the *level of terms.* Terms (or expressions) can never appear alone, but only as part of an object definition. In Coq, the terms are CIC λ-expressions, i.e. variables (encoded as DeBrujin indexes), λ-abstractions and applications, product types and sorts, augmented with a system of inductive types in the spirit of the ones of Martin-Löf, comprising (co)inductive types and constructors, case analysis operators and inductive and co-inductive function definitions. The whole level is extremely dependent from the logical framework.

The second level, that uses the previous one to encode both bodies and types, is the one of *objects.* Every object is stored into a different file. The files are structured into directories that corresponds to sections in Coq, i.e. delimiters of the scope of a variable. Sections are also used in Coq to structure a large theory into sub-theories. In HELM, the former usage is retained, while theories are described in another way (see the third level).

Constants (definitions/theorems/axioms) Constant objects of Coq are used to represent definitions, theorems and also axioms. Definitions and theorems are syntactically identical and have a body and a type. The only difference is semantical: theorems are usually opaque (only their type is used in CIC terms) because of proof-irrelevance, while definitions are transparent (their body is substituted in their occurrences in other terms during type-checking). Axioms, instead, are constants with a type but without a body. We choose, during extraction, to discriminate only axioms from definitions and theorems, that become indistinguishable once extracted. We leave to the next level (that of theories) the responsibility of "marking" non-axiom constants as theorems, definitions, lemmas, facts, ...

Variables Variables have only a type and not a body. A variable behaves like an axiom inside the section where it is defined and as a parameter when referring to another object of that section. Hence, sections are used to delimit the scope of variables.

(Co)Inductive Definitions In Coq, blocks of mutual (co)inductive definitions can also be defined. Each definition inside such a block has a name, a type (called arity in Coq) and a possibly empty list of constructors. Each constructor has a name and a type. A simple example is the inductive type of natural numbers whose name is nat, whose type is Set and whose constructors are O of type nat and S of type (nat $\rightarrow$ nat).

Blocks, as constants, could depend on variables; the list of variables (parameters) on which all the definitions in the block depend is also exported from Coq.

Proof in progress We also choose to export unterminated proofs. As terminated theorems, an unterminated proof has a name, a type and a body; moreover, it has also a list of conjectures on which the body depends. Each conjecture has a type but not a body: to end the proof you must provide a body for each conjecture.

An example of an object file describing a constant (a theorem) can be found in appendix A.

The last level is the *level of theories* which is completely independent from the particular logical framework. In our idea, a theory is a (structured) mathematical document containing objects taken almost freely from different sections. Writing a new theory should consist in developing new objects and assembling these new objects and older ones into the mathematical document. It is during the creation of a theory that objects must also be assigned the particular, semantical meaning used to classify them, for example into lemmas, conjectures, corollaries, etc. Theories, that are described in different XML files, do not include the objects directly, but refers to them via their URIs.

Theory files also have sections delimiting the scope of variable declarations: to include (a reference to) a definition D depending on a variable V when both reside in a section (directory) R, you must open, inside the theory file, a section referring to R and put inside it the references to V and R. A very small example of a theory file will clarify the above statement:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Theory SYSTEM "http://www.cs.unibo.it/helm/dtd/maththeory.dtd">
<Theory uri="cic:/Coq/Init/Logic">
 <!-- Require Export Datatypes -->
 <DEFINITION uri="True.ind"/>
 <DEFINITION uri="False.ind"/>
```

```
 <DEFINITION uri="not.con"/>
 <SECTION uri="Conjunction">
  <DEFINITION uri="and.ind"/>
  <VARIABLE uri="A.var"/>
  <VARIABLE uri="B.var"/>
  <THEOREM id="id1" uri="proj1.con"/>
  <THEOREM id="id2" uri="proj2.con"/>
 </SECTION>
 <SECTION uri="Disjunction">
  <DEFINITION uri="or.ind"/>
 </SECTION>
</Theory>
```

All the URIs, but that of `Theory`, are relative URIs; so, the absolute URI of `id1` is "`cic:/Coq/Init/Logic/Conjunction/id1`". In the example you can also see the usage of sections to bound the scope of variables: the scope of `A` and `B` is the section `Conjunction`.

It is important to note that at the theory level, sections are not used to structure the document into, for instance, chapters, paragraphs and so on; many kind of (XML) markup languages have just been developed to do so. Accordingly to the spirit of XML, our theory markup will be freely and modularly intermixed with other kinds of markup, such as XHTML[5]; so, our language will play for mathematical theories the same role of MathML for mathematical expressions and SVG[6] for vectorial graphics. The added value of using the theory level (instead of directly embedding the object markup) is that, while enriching the semantics of the objects of the previous level, it could also be used to enforce some constraints as, for example, on the scope of variables or on the links between theorems and lemmas.

The module is full working and has been used to export the whole library provided with the Coq System, yielding about 64 Mb of XML (2 Mb after compression). All the obtained XML documents are valid with respect to the DTDs developed for the three levels; we cannot show the DTDs here because of lack of space.

4.2 Auxiliary Information & Metadata

In this paragraph we address the problem of the association of additional and possibly redundant information to the one exported using the Coq module. The purpose of this added information is to enable or facilitate specific functionalities such as rendering, searching, indexing and so on. A simple observation suggests that such information could be naturally associated either to the whole object definition (e.g. the name of the author of a theorem) or to a particular node of a λ-term (e.g. the tactic used to generate it or an informal description of its meaning). So, we can easily store the additional information in a distinct XML file and use XLink technology to relate it to the appropriate XML element in the corresponding logical-level document. Moreover, in the specific case of metadata, we can benefit from the Resource Description Framework (RDF, [10] [11]), which provides a general model for representing metadata as well as a syntax for encoding and exchanging this metadata over the Web. In several cases, this meta-information can be extracted from the document, for instance by means of XSL Transformations. It is important to note that, in this approach, no modification at all is required to the DTDs or to the source document. As a side effect, an application can consult just the XML files holding the information it really needs, without having to parse, check for consistency and finally ignore non-interesting information.

[5] http://www.w3.org/TR/xhtml1
[6] http://www.w3.org/TR/SVG

An example of additional information that we have considered are types for λ-term sub-expressions, here called inner-types. λ-terms are used in logical frameworks to encode proofs, with each node corresponding to the application of a logical rule; as showed by Yann Coscoy [16], though, in order to give a natural description of proofs, inner-types are as important as terms. So, for example, to render the proof $\lambda a : A.\lambda ab : A \to B.\lambda bc : B \to C.(bc(ab\ a))$ of $A \to (A \to B) \to (B \to C) \to C$, we need to know that the thesis of the subproof $(ab\ a)$ is B, allowing a rendering as "First we prove B by $(ab\ a)$; hence we get A using bc"[7].

Inner-types provide information which is redundant with respect to the one already exported: they can be inferred from the XML representation. We have provided an implementation in XSLT that we have merged with the prototype stylesheets for the natural-language rendering, but we have then dropped this approach due to performance problems (XSLT is an interpreted language for which there does not exist now a really efficient interpreter) and also because the same type-inference operation was done several times to get the same results. Moreover, from a logical point of view, type-inference should not be part of rendering at all. So, waiting for a stand-alone implementation of a type-inference engine, we have now modified the exporting module for Coq V7 to add to each XML exported file a second one holding the inner-types for the sub-terms in it.

5 Proof-Checker

In order to verify that all the needed core information was exported from Coq, we have developed a stand-alone proof-checker for CIC objects, similar to the Coq one, but fairly simpler and smaller thanks to its independence from the proof engine. The proof-checker is now almost finished, only lacking the management of universes. It is the first example of a tool working directly on the XML encoding.

With respect to other proof-checkers (as the one of Coq), it is fairly standard but for the peculiar management of the environment: usually, proof-checkers are used to check whole theories, i.e. sequence of definitions or proofs. Each time a definition is checked, it is added to the environment and then it is used in subsequent type-checking. So, every theorem is always checked with the same, statically defined environment [8]. Our proof-checker, instead, is also used to check single objects in an environment that could not yet contain the definitions required (e.g. the empty environment). In this case, the environment (a cache, actually) is built "on-demand" during the type-checking of the object: every time a reference to another object not present in the environment is found, the type-checking is interrupted, processing the new object first. Checks are introduced in order to avoid cycles in the definitions, corresponding to an inconsistent environment. In order to make the user understand the strange behaviors described in note 8, that could in theory appear more often in our model, we advocate a way for the user to ask the system what is the inferred universe level of each type.

6 XSL Transformations and MathML

XSLT [9] is a language for transforming XML documents into other XML documents. In particular, a stylesheet is a set of rules expressed in XSLT to transform the tree representing a XML document (the Document Object Model, DOM [3]) into a result tree. When a pattern

[7] In appendix B is reported a proof in which the inner-types have been exploited to get a raw natural-language rendering.

[8] This is almost true: a user is free to load two theories in any order, changing in this way the environment used during the second type-checking. Sometimes this can lead to strange behaviors, i.e. two theories that are correct if loaded alone could be inconsistent if loaded together. This phenomenon is due to the inference of the universe level and in particular to the creation of cycles between the universe level constraints.

is matched against elements in the source tree, the corresponding template is instantiated to create part of the result tree. In this way the source tree can be filtered and reordered, and arbitrary structure can be added. A pattern is an expression of XPath [8] language, that allows to match elements according to their values, structure and position in the source tree.

XSLT is primarily aimed to associate a *style* to a XML document, generating formatted documents suitable for rendering purposes. Once XML is chosen as the data description language to encode the mathematical information, it is quite natural to use stylesheets as the standard mechanism to automatically generate, starting from a XML mathematical document, several alternative notations and encodings.

A natural encoding for mathematics on the Web is MathML [12], an instance of XML for describing the notation of a mathematical expression, capturing both its structure and content. MathML has, roughly, two categories of markup elements: the presentation markup, which can be used to describe the layout structure of mathematical notation, and the content markup, whose aim is to provide an explicit encoding of the *underlying* mathematical structure of an expression.

Although the target of MathML is the encoding of expressions (so it cannot describe mathematical objects and documents), the MathML Presentation Markup can be considered as a privileged rendering format, providing a standard on the web already implemented by several applications, and especially being a quite powerful output format.

Also, the choice of MathML content markup as an intermediate representation between the logic-dependent representation of the mathematical information and its presentation is justified by several reasons:

- Even if the content markup is restricted to the encoding of a particular set of formulas (the ones used until the first two years of college in the United States), it is extensible and flexible[9].
- Passing through this semi-formal representation will 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). Moreover several stylesheets can be written for this single intermediate language to generate specific rendering formats.
- The characteristic of portability of MathML content markup can be exploited, for instance when cutting and pasting terms from an application to another.
- MathML content markup can capture the usual *informal* semantics of well-known operators, as for example the disjunction, marking them with the corresponding content elements (e.g. `or`). Their different formal content is preserved by means of backward pointers to the low level specification.

A valid alternative to MathML content is provided by OpenMath[10] which is a standard for representing the semantic meaning of mathematical objects, in particular providing a way to describe the semantics of new defined operators (`csymbols` in MathML). To this purpose, special XML files called Content Dictionaries (CD) are provided. Note, however, that without a formal encoding for definitions, advocated in our project, the information put in content dictionaries could only be represented by informal descriptions in natural language ("plus: The symbol representing an n-ary commutative function plus."[11]) or semi-formal ones that are definitions given in OpenMath that relies on other definitions already informally given. So, OpenMath has no added value for our purposes than MathML content. On the contrary, OpenMath Content Dictionaries could really benefit by our formal descriptions.

[9] The most important element for extensibility purposes is `csymbol`, defined for constructing a symbol whose semantics is not part of the core content elements provided by MathML, but defined externally.

[10] http://www.openmath.org/.

[11] http://www.openmath.org/cd/arith1.ocd

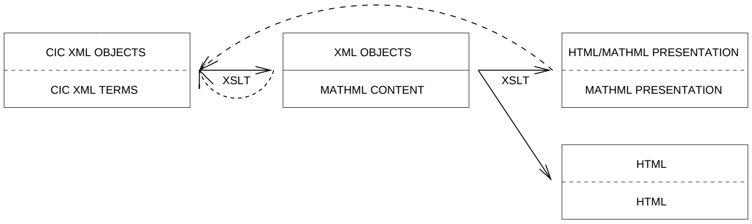

Figure 2: Transformations on the first two levels of CIC XML files: the backward arrows represent links from the content and presentation files to the corresponding CIC XML files.

As you can see in Figure 2, there are two phases of stylesheets application: the first generates the content representation from the CIC XML one; the second starting from this intermediate representation generates two (and eventually several others) possible output formats, either the MathML presentation markup or the HTML markup. Note that MathML content can only describe CIC terms (expressions) and so we have had to add a second language to describe CIC objects in the intermediate step. Every obtained document is valid with respect to the corresponding DTD developed for it; in particular, we use the DTD recommended by the last MathML Recommendation.

This is an example of content markup[12]:

```
<m:apply>
 <m:csymbol>app</m:csymbol>
 <m:ci definitionURL="cic:/Coq/Init/Logic/Conjunction/and_ind.con">and_ind</m:ci>
 <m:ci>A</m:ci>
 <m:ci>B</m:ci>
 <m:ci>A</m:ci>
 <m:lambda>
  <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
  <m:lambda>
   <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
   <m:ci>H0</m:ci>
  </m:lambda>
 </m:lambda>
 <m:ci>H</m:ci>
</m:apply>
```

During the realization of the content stylesheets, we have had to face and solve several problems connected to the MathML specification:

- Objects, in general, cannot be considered as terms. Hence, we have added to the MathML content markup a XML level to describe the level of objects (the mixing is possible using the W3C namespaces [5]). Anyway, this language is still dependent on the particular logical framework.
- MathML lacks any standard semantics associated to its content elements, so we need to semantically describe not only entities defined in CIC, as the conjunction, but also the primitive CIC operators, as the application or the λ-abstraction. To preserve their formal semantics, we keep pointers to the XML files of their CIC definitions.
- With regard to most CIC primitive operators, there are no specific MathML content markup elements. To solve this problem we use the `csymbol` element. In particular,

[12]This fragment belongs the example of content file in Appendix A.

we have chosen to introduce also a `csymbol` for the CIC application (see the example above) instead of using the content element `apply`. In fact the MathML application is the general way of building up a mathematical expression, and so it is different from the CIC application. So we want to make a clean semantical distinction between the logical application between two terms, and the "application" of some operators (like the sin function) to its arguments.

As you can see in Figure 2, we produce MathML content and presentation in two distinct steps. The only way to combine and link together content and presentation in compliance to the MathML specification consists of using the `semantics` element. This content element is quite ambiguous, a kind of "bridge" between content and presentation; moreover, it is currently ignored by all the browsers supporting MathML. For us, the next natural improvement will consist of having content and the associated presentations in different files, one for the content expression and one for each presentation. Then we need to relate a presentation expression and subexpressions to the respective content expression and subexpressions. This can be achieved in a standard way using the machinery of XLink and XPointer [6, 7].

The above solution has been also exploited for the implementation of the links of Figure 2 for linking the content and presentation markup to the corresponding source CIC XML terms. In this way the user can browse the MathML presentation and also modify it: the changes will have effect on the corresponding CIC XML file.

An example of MathML presentation markup generated after the second phase is[13]:

```
<m:mrow>
 <m:mo stretchy="false">(</m:mo>
 <m:mi>and_ind</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>A</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>B</m:mi>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>A</m:mi><m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mrow>
  <m:mo color="Red">&lambda;</m:mo>
  <m:mi>H0</m:mi>
  <m:mo>:</m:mo>
  <m:mi>A</m:mi>
  <m:mo>.</m:mo>
  <m:mrow>
   <m:mo color="Red">&lambda;</m:mo>
   <m:mi>H1</m:mi>
   <m:mo>:</m:mo>
   <m:mi>B</m:mi>
   <m:mo>.</m:mo>
   <m:mi>H0</m:mi>
  </m:mrow>
 </m:mrow>
 <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
 <m:mi>H</m:mi>
 <m:mo stretchy="false">)</m:mo>
</m:mrow>
```

To generate the presentation markup from the corresponding content markup, we use, among others, a stylesheet, compliant with the last specification of MathML, written by Igor

[13] This fragment belongs the the example of content file in Appendix A.

Rodionov[14]. This stylesheet, written in collaboration with the MathML Working Group, transforms MathML content markup in MathML presentation one.

We have had to solve several problems regarding the presentation output:

- We have had to associate an output to every object and to every `csymbol` defined in the content phase.
- We have modified the MathML stylesheet to implement the policy of line-breaking and alignment for long terms: our choice consists of using tables made of multiple rows. The `mtable` element is specifically designed to arrange expressions in a bi-dimensional layout and in particular it provides a set of related elements and attributes to achieve proper alignment and line-breaking. Our policy consists of breaking expressions only in specific points and only when the row length exceeds a threshold (that could make the reading difficult), generating tables for alignment purposes. On the contrary, regarding the HTML output format, we have been forced to implement a spacing policy, because of the nature of HTML tables, which cannot be aligned to a row of other tables.

As we have said above, MathML is not the only format exploited: another presentation format is HTML, due to the wide-spreading of browsers for it and its native hypertextual nature. Thanks to the modular architecture (see Figure 2), many other formats could be added too.

Finally, note that the general architecture gives the possibility of applying stylesheets of arbitrary complexity. For example, we are developing stylesheets to render proofs in natural language, in the spirit of the work of Yann Coscoy [17].

We will exploit the same modular architecture of the object level at the level of theories. At this level we can use the same presentation formats of the previous levels; on the contrary, there is no standard language for describing theories at the content level. So we will develop a new (XML) language that will be largely independent from the specific foundational dialect and could aspire to play the same role MathML plays for mathematical expressions. An already existent corresponding proposal for OpenMath is OMDoc [13]. A possibility could consist of achieving the integration of these two proposals, giving a common standard way to represent the structure of a mathematical document.

7 Interfaces to HELM

Two of the main goals of the project are the easiness in augmenting and browsing the library:

1. Every user with a small amount of HTTP or FTP space should be able to publish a document.
2. Every user with a common browser should be able to browse the library.

To fulfill these aims, we must face the actual state of technology:

1. Currently, almost all of the Internet users have a web space, but usually without being allowed to run any kind of program on the server, even simple CGIs. So no intelligence can be put on the publisher side.
2. The browser technology is rapidly evolving in such a way that we can expect in a few time to have browsers able to understand MathML and, probably, even to apply XSLT stylesheets. At the moment, though, if we require the browser to be standard, then we have to put the intelligence on the other side, i.e. on the server.

[14] Computer Science Department of the University of Western Ontario, London, Canada.

Therefore, where can we put the intelligence? A first answer is the creation of *presentation sites* able to retrieve documents from distribution sites, process them (e.g. applying stylesheets) and return them to the users in the user requested format.

In a first prototype of presentation site we relied on the capabilities of Cocoon, a XML server-based web-publishing framework, for the processing of the requested documents.

Recently we adopted a more modular and more reusable solution, consisting in having a completely separate component, a servlet indeed, for the application of stylesheets. This component is designed to receive HTTP requests of stylesheet application, sending the result of the transformation as the reply of the request. In this way, ordinary browsing tools can access the processed documents simply by invoking the servlet with a proper URL containing both a reference to the source document to be processed, the list of stylesheets to be applied and possibly a sequence of parameters. In this way the number and order of the stylesheets can be changed at any invocation and their behaviour may be affected by the value of the parameters passed by the client. Moreover, the documents, the stylesheets and the DTDs are not constrained to reside on the same host of the processor, thus over-passing the greatest limitation of the Cocoon-based framework.

An on-line version of the standard library of the second beta-release of Coq V7 can be found on-line at the address `http://phd.cs.unibo.it/helm/library/`. It is important to note that each document is rendered either to HTML or to MathML presentation on-the-fly.

Though this solution is perfect for browsing and doing simple elaborations, it gives the user too strict interaction possibilities, which are required for more complex tasks (as the creation of new theories, for example). Hence, more advanced interfaces with such capabilities are required. These interfaces must be run on the user's machine and should, at least, be able to provide all the processing functionalities of the presentation servers. At the same time, they should also overcome the limitations of standard browsers through the addition of new interaction possibilities.

Since our preferential browsing language will be MathML, our interface should at least be able to render its presentation markup. Unfortunately, there are no satisfactory implementations available yet. Moreover, we also need to interact with the MathML rendered files, for example for editing. Not only forms of interaction with this kind of markup have never been proposed before, but we also need to reflect the changes on the source files of the XSLT rendering transformations. This has lead us to the development of a brand new engine with rendering and editing capabilities for documents embedding MathML presentation markup. The widget is also able to export rendered MathML documents to Postscript, as the one presented in appendix B. This engine is designed to be stand-alone and is freely available as a Gtk widget[15].

We have integrated our widget, the proof-checker and the external XSLT processor into a minimal interface that we are going to extend with editing functionalities.

8 The model of distribution

Mathematical documents have some peculiar properties. First of all a mathematical document should be immutable: the correctness of a document A that refers to a document B can be guaranteed only if B does not change. Notwithstanding this, new versions of a mathematical document could be released (for example if a conjecture is actually proved). Secondly, a user cannot be forced to retain a copy of his document forever, even if other documents refer to it. So, it should be possible for everyone to make a copy of a document and also distribute

[15]The widget is now a package of the forthcoming Debian distribution.
See `http://www.cs.unibo.it/helm/mml-widget/` for further details.

it. When more than a copy is available, the user should be able to download it from any suitable server (for example, from the nearest one). This implies that documents could not be referred to via URLs, but only with logical names in the form of URIs. A particular naming policy should then be adopted to prevent users to publish different documents under the same URI.

To fulfill these requirements, we have adopted almost the same model of distribution of the Debian packaging system APT[16] which has similar requirements (a package could not be modified, but only updated, it is available on different servers, could be downloaded from the user preferred server).

Every document is identified by an URI. "`cic:/Coq/Init/Datatypes/nat.ind`" is an example of such an URI that references an inductive definition (".ind") in the subsection `Datatypes` of the subsection `Init` of the section `Coq`.
Similarly, the URI "`theories:/Coq/Init/Datatypes.theory`" refers to the mathematical theory named `Datatypes` located in the subdirectory `Init` of the directory `Coq`.

In order to translate the URI to a valid URL, a particular tool, named *getter*, is needed. It takes in input an ordered list of servers and an URI and returns the URL of the document on the first server in the list providing it. In order to know which documents a server provides, each server publishes a list of the URIs of its documents with the respective URLs.

In order to flexibly allow communication between the components of HELM architecture, the interface provided by the getter is the one of an HTTP daemon, pre-dating the ideas of the SOAP working draft [18]: when a document is required, its logical name is resolved in an URL, the document is downloaded, it is processed if needed[17] and finally it is returned to the client and it is possibly locally stored for caching purposes. If the machine hosting the getter is also a distribution site, then, once cached, a document could also be added to the list of documents the server exports. In such a way, often referred to or simply interesting documents fast widespread over the net, downloading times are reduced and the author can freely get rid of his copy of the document if he does not need it any more.

This architecture imposes no constraints on the naming policy: up to now we have not chosen or implemented one yet. To face the issue, one possibility can be the choice of having a centralized naming authority, even if other more distributed scenarios will surely be considered.

9 Further Developments

We are soon going to develop:

Tools for indexing and retrieval of mathematical documents, based on meta-data specified in the Resource Description Framework (RDF, see [10]). RDF uses XML to define a foundation for processing meta-data, complements XML and provides interoperability between applications that exchange machine-understandable information on the Web.

Tools for the (re)annotation of mathematical objects and terms: the intuitive meaning of these entities is usually lost in their description in a logical framework. Even their automatically extracted presentations in a natural language are often unsatisfactory, being quite different from the typical presentation in a book. We believe that a feasible solution is giving the user the possibility of enriching terms with annotations given in an informal, still structured language.

[16] http://www.debian.org

[17] For example, typical processing are deflating compressed files or the resolution of relative URIs contained in the document to absolute ones

Tools to provide functionalities missing from monolithic proof assistants, such as proof-extraction of proof-editing.

10 Conclusions

In this paper we have presented the current status of the HELM project, whose aim is to exploit the potentiality of XML technology for the creation and maintenance of large repositories of structured, content oriented information offered by Logical Frameworks. In this way we meet the ultimate goal of the Semantic Web, that is to allow machines the sharing and exploitation of knowledge in the Web way, i.e. without central authority, with few basic rules, in a scalable, adaptable, extensible manner.

Our ultimate goal is the extension of the library to other logical frameworks and systems. This will also be an important test bench for the whole architecture.

Another fundamental improvement would be the development of new modular proof engines, supporting step-by-step informal annotations on proofs in natural language (see [14] for a deeper discussion of this topic).

References

[1] 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.

[2] Berners-Lee, T., "Universal Resource Identifiers in WWW", RFC 1630, CERN, June 1994.

[3] Document Object Model (DOM) Level 2 Specification. Version 1.0, W3C Recommendation, 13 November 2000.
`http://www.w3.org/TR/2000/CR-DOM-Level-Core/`
`http://www.w3.org/TR/DOM-Level-2-Views/`
`http://www.w3.org/TR/DOM-Level-2-Events/`
`http://www.w3.org/TR/DOM-Level-2-Style/`
`http://www.w3.org/TR/DOM-Level-2-Traversal-Range/`

[4] Extensible Markup Language (XML) (Second Edition). Version 1.0. W3C Recommendation, 6 October 2000.
`http://www.w3.org/TR/REC-xml`

[5] Namespaces in XML. W3C Recommendation, 14 January 1999.
`http://www.w3.org/TR/REC-xml-names/`

[6] XML Linking Language (XLink). Version 1.0. W3C Proposed Recommendation, 20 December 2000.
`http://www.w3.org/TR/xlink/`

[7] XML Pointer Language (XPointer). Version 1.0. W3C Working Draft (Last Call), 8 January 2001.
`http://www.w3.org/TR/xptr`

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

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

[10] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[11] Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000. http://www.w3.org/TR/rdf-schema/

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

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

[14] Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I., "XML, Stylesheets and the re-mathematization of Formal Content", Department of Computer Science, Bologna, Italy, May 2000.

[15] B. Barras et al., "The Coq Proof Assistant Reference Manual, version 6.3.1", http://pauillac.inria.fr/coq/

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

[17] Y.Coscoy. "Explication textuelle de preuves pour le Calcul des Constructions Inductives", Phd. Thesis, Université de Nice-Sophia Antipolis, 2000.

[18] Simple Object Access Protocol (SOAP). Version 1.1. W3C Note, 8 May 2000.
http://www.w3.org/TR/SOAP/.

A

Cic Xml file:

```
<!DOCTYPE HTML SYSTEM "http://www.cs.unibo.it/helm/dtd/cic.dtd">
<Definition name="proj1" params="0: A B">
  <body>
   <LAMBDA>
     <source>
       <APPLY>
         <MUTIND notype="0" uri="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
       </APPLY>
     </source>
     <target binder="H">
       <APPLY>
         <CONST uri="cic:/Coq/Init/Logic/Conjunction/and_ind.con"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
         <VAR reluri="0,A"/>
         <LAMBDA>
           <source><VAR reluri="0,A"/></source>
           <target binder="H0">
             <LAMBDA>
               <source><VAR reluri="0,B"/></source>
               <target binder="H1"><REL binder="H0" value="2"/></target>
             </LAMBDA>
           </target>
         </LAMBDA>
         <REL binder="H" value="1"/>
       </APPLY>
     </target>
   </LAMBDA>
  </body>
  <type>
   <PROD>
     <source>
       <APPLY>
         <MUTIND notype="0" uri="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
         <VAR reluri="0,A"/>
         <VAR reluri="0,B"/>
       </APPLY>
     </source>
     <target><VAR reluri="0,A"/></target>
   </PROD>
  </type>
</Definition>
```

Corresponding content file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Definition name="proj1" xmlns:m="http://www.w3.org/1998/Math/MathML">
  <Params>0: A B</Params>
  <body>
   <m:math>
    <m:lambda>
     <m:bvar>
      <m:ci>H</m:ci>
      <m:type>
       <m:apply>
        <m:and definitionURL="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
        <m:ci>A</m:ci>
        <m:ci>B</m:ci>
       </m:apply>
      </m:type>
     </m:bvar>
     <m:apply>
      <m:csymbol>app</m:csymbol>
      <m:ci definitionURL="cic:/Coq/Init/Logic/Conjunction/and_ind.con">and_ind</m:ci>
      <m:ci>A</m:ci>
      <m:ci>B</m:ci>
      <m:ci>A</m:ci>
      <m:lambda>
```

```
      <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
      <m:lambda>
       <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
       <m:ci>H0</m:ci>
      </m:lambda>
     </m:lambda>
     <m:ci>H</m:ci>
    </m:apply>
   </m:lambda>
  </m:math>
 </body>
 <type>
  <m:math>
   <m:apply>
    <m:csymbol>arrow</m:csymbol>
    <m:apply>
     <m:and definitionURL="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
     <m:ci>A</m:ci>
     <m:ci>B</m:ci>
    </m:apply>
    <m:ci>A</m:ci>
   </m:apply>
  </m:math>
 </type>
</Definition>
```

Corresponding presentation file:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/strict.dtd">
<m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
 <m:mtable align="baseline 1" columnalign="left" equalrows="false">
  <m:mtr>
   <m:mtd><m:mrow><m:mtext>DEFINITION proj1() OF TYPE</m:mtext></m:mrow></m:mtd>
  </m:mtr>
  <m:mtr>
   <m:mtd>
    <m:mrow>
     <m:mphantom><m:mtext>__</m:mtext></m:mphantom>
     <m:semantics>
      <m:mrow>
       <m:mo stretchy="false">(</m:mo>
       <m:mrow>
        <m:mi>A</m:mi>
        <m:mo><mchar name="wedge"></mchar></m:mo>
        <m:mi>B</m:mi>
       </m:mrow>
       <m:mo color="Blue">&rarr;</m:mo>
       <m:mi>A</m:mi>
       <m:mo stretchy="false">)</m:mo>
      </m:mrow>
      <m:annotation-xml encoding="MathML">
       <m:apply>
        <m:csymbol>arrow</m:csymbol>
        <m:apply>
         <m:and definitionurl="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
         <m:ci>A</m:ci>
         <m:ci>B</m:ci>
        </m:apply>
        <m:ci>A</m:ci>
       </m:apply>
      </m:annotation-xml>
     </m:semantics>
    </m:mrow>
   </m:mtd>
  </m:mtr>
  <m:mtr><m:mtd><m:mrow><m:mtext>AS</m:mtext></m:mrow></m:mtd></m:mtr>
  <m:mtr>
   <m:mtd>
    <m:mrow>
     <m:mphantom><m:mtext>__</m:mtext></m:mphantom>
     <m:semantics>
      <m:mrow>
       <m:mo color="Red">&lambda;</m:mo>
       <m:mi>H</m:mi>
```

```xml
        <m:mo>:</m:mo>
        <m:mrow>
         <m:mi>A</m:mi>
         <m:mo><mchar name="wedge"></mchar></m:mo>
         <m:mi>B</m:mi>
        </m:mrow>
        <m:mo>.</m:mo>
        <m:mrow>
         <m:mo stretchy="false">(</m:mo>
         <m:mi>and_ind</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>A</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>B</m:mi>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>A</m:mi><m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mrow>
          <m:mo color="Red">&lambda;</m:mo>
          <m:mi>H0</m:mi>
          <m:mo>:</m:mo>
          <m:mi>A</m:mi>
          <m:mo>.</m:mo>
           <m:mrow>
            <m:mo color="Red">&lambda;</m:mo>
            <m:mi>H1</m:mi>
            <m:mo>:</m:mo>
            <m:mi>B</m:mi>
            <m:mo>.</m:mo>
            <m:mi>H0</m:mi>
           </m:mrow>
         </m:mrow>
         <m:mphantom><m:mtext>_</m:mtext></m:mphantom>
         <m:mi>H</m:mi>
         <m:mo stretchy="false">)</m:mo>
        </m:mrow>
       </m:mrow>
       <m:annotation-xml encoding="MathML">
        <m:lambda>
         <m:bvar>
          <m:ci>H</m:ci>
          <m:type>
           <m:apply>
            <m:and definitionurl="cic:/Coq/Init/Logic/Conjunction/and.ind"/>
            <m:ci>A</m:ci><m:ci>B</m:ci>
           </m:apply>
          </m:type>
         </m:bvar>
         <m:apply>
          <m:csymbol>app</m:csymbol>
          <m:ci definitionurl="cic:/Coq/Init/Logic/Conjunction/and_ind.con">and_ind</m:ci>
          <m:ci>A</m:ci>
          <m:ci>B</m:ci>
          <m:ci>A</m:ci>
          <m:lambda>
           <m:bvar><m:ci>H0</m:ci><m:type><m:ci>A</m:ci></m:type></m:bvar>
           <m:lambda>
            <m:bvar><m:ci>H1</m:ci><m:type><m:ci>B</m:ci></m:type></m:bvar>
            <m:ci>H0</m:ci>
           </m:lambda>
          </m:lambda>
          <m:ci>H</m:ci>
         </m:apply>
        </m:lambda>
       </m:annotation-xml>
      </m:semantics>
     </m:mrow>
    </m:mtd>
   </m:mtr>
  </m:mtable>
</m:math>
```

B

Coq Term as rendered by Coq:

```
incl_add =
[U:Type;
 A,B:(Ensemble U);
  x:U;
   H':(Included U A B);
    x0:U;
     H'0:(In U (Add U A x) x0)]
      (or_ind (In U A x0) x==x0 (In U (Add U B x) x0)
         [H'2:(In U A x0)](Add_intro1 U B x x0 (H' x0 H'2))
    [H'2:(x==x0)]
        (eqT_ind U x [u:U](In U (Add U B x) u) (Add_intro2 U B x) x0 H'2)
   (Add_inv U A x x0 H'0))
        : (U:Type; A,B:(Ensemble U); x:U)
        (Included U A B)->(Included U (Add U A x) (Add U B x))
```

Coq Term as rendered by the HELM interface:

DEFINITION incl_add(0: U) OF TYPE
$\forall\, A : (\text{Ensemble } U)$
$.\forall\, B : (\text{Ensemble } U)$
$.\forall\, x : U.(A \subseteq B \rightarrow A \bigcup \{x\} \subseteq B \bigcup \{x\})$
AS
$\lambda\, A : (\text{Ensemble } U)$
$.\ \lambda\, B : (\text{Ensemble } U)$
$.\ \lambda\, x : U$
$.\ \lambda\, \mathrm{H'} : A \subseteq B$
$.\ \lambda\, x_0 : U$
$.\ \lambda\, \mathrm{H'}_0 : x_0 \in A \bigcup \{x\}$
$.(h_1)\ \lambda\, \mathrm{H'}_1 : x_0 \in A \vee x = x_0$
.Consider $\mathrm{H'}_1$
We prove $x_0 \in B \bigcup \{x\}$ by cases:
$*\ \lambda\, \mathrm{H'}_2 : x_0 \in A$
$.(\mathrm{H'}\ \ x_0\ \ \mathrm{H'}_2)$
we proved $x_0 \in B$
$(\text{Add\_intro}_1\ \ U\ \ B\ \ x\ \ x_0\ \text{ previous})$
we proved $x_0 \in B \bigcup \{x\}$
$*\ \lambda\, \mathrm{H'}_2 : x = x_0$
$.(\text{Add\_intro}_2\ \ U\ \ B\ \ x)$
we proved $x \in B \bigcup \{x\}$
Rewrite x with x_0 by $\mathrm{H'}_2$
we proved $x_0 \in B \bigcup \{x\}$
we proved $x_0 \in B \bigcup \{x\}$
we proved $(x_0 \in A \vee x = x_0 \rightarrow x_0 \in B \bigcup \{x\})$
$(h_2)(\text{Add\_inv}\ \ U\ \ A\ \ x\ \ x_0\ \ \mathrm{H'}_0)$
we proved $x_0 \in A \vee x = x_0$
$(h_1\ \ h_2)$
we proved $x_0 \in B \bigcup \{x\}$
we proved $\forall\, A : (\text{Ensemble } U)$
$.\forall\, B : (\text{Ensemble } U)$
$.\forall\, x : U$
$.(A \subseteq B \rightarrow \forall\, x_0 : U.(x_0 \in A \bigcup \{x\} \rightarrow x_0 \in B \bigcup \{x\}))$