An Interactive Algebra Course with Formalised Proofs and Definitions

Andrea Asperti[1], Herman Geuvers[2], Iris Loeb[2], Lionel Elie Mamane[2], and Claudio Sacerdoti-Coen[3]

[1] Dept. of Comp. Science, University of Bologna, Italy
[2] ICIS, Radboud University Nijmegen, NL
[3] Project PCRI, CNRS, École Polytechnique, INRIA, Université Paris-Sud, France

Abstract. We describe a case-study of the application of web-technology (Helm [2]) to create web-based didactic material out of a repository of formal mathematics (C-CoRN [5]), using the structure of an existing course (IDA [4]). The paper discusses the difficulties related to associating notation to a formula, the embedding of formal notions into a document (the "view"), and the rendering of proofs.[4]

1 Introduction

One of the aims of the recently concluded European IST Project MoWGLI was the development of a suitable technology supporting the creation of web-based didactic material out of repositories of formal mathematical knowledge.

In particular, the validation activity reported in this paper consists of the application of the Helm [2] technology, developed at the University of Bologna, to the C-CoRN [5] repository of constructive mathematics of the University of Nijmegen and aiming at the creation of an interactive algebra course.

The Helm system [2] provides tools and techniques for displaying formalised mathematics on the web. It uses XML technology for rendering repositories of formal mathematics, supporting hyperlinks, browsing and querying functionalities, as well as a sophisticated stylesheet mechanism for the notational reconstruction of the symbolic content.

The C-CoRN system [5] is a repository of constructive mathematics, formalised in Coq, covering a considerable body of basic algebra and analysis.

Potentially, Helm and C-CoRN produce a large body of formalised mathematics, which can be viewed and browsed through standard web tools. In order to exploit this potentiality, Helm must be suitably *instantiated* to the particular case of C-CoRN. This instantiation essentially takes place at two levels:

notational defining (directly or indirectly) a set of XSLT transformations providing the required notational rendering for the formal notions coded in C-CoRN;

[4] This research was supported by the European Project IST-33562-MoWGLI

structural providing the didactic organisation and the natural language glue of the course notes

This paper is a report of the work. The final result is available at `http://helm.cs.unibo.it/`. The structure of the paper is the following. Sect. 2 and 3 introduce respectively the Helm system and the C-CoRN repository; in Sect. 4 we discuss the association of notation to a formula and Sect. 5 is about documents as views; the rendering of proofs is addressed in Sect. 6. Finally we draw some general conclusions about this validation activity.

2 Helm

The process of transforming a Coq proof to a XHTML or MathML-Presentation proof is shown in Fig. 1. The process is essentially split in two parallel pipelines, respectively dealing with *proof objects*, i.e. single mathematical items such as theorems, definitions, examples and so on, and *views* that are structured collections of (links to) objects, possibly intermixed with text and pictures.

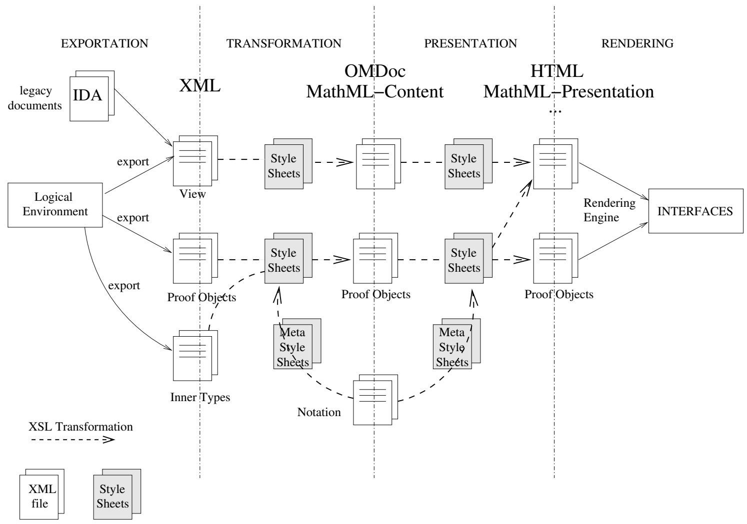

Fig. 1. Transformation process in Helm

From the Coq files, the lambda-term is exported to an XML-language, CICML. Similarly, from scripts we export a sort of minimal, canonical view, that is essentially the index of theorems in the order they have been defined by the user, plus some special comments possibly added to the script. The XML exportation module is currently a standard component of the Coq distribution.

An alternative, simple way to produce a view is by starting from some legacy documents, for instance using a traditional latex to HTML converter (links to the repository have to be added manually). In Section 5 we shall describe in detail this job, adopting, as a view, the HTML material of the Interactive Algebra Course [4] on algebra of the Eindhoven University of Technology (NL).

The raw XML encoding used by Helm to store the information in the repository is transformed into a presentational markup by a suitable set of stylesheets. This transformation process is split, for modularity reasons, in two main parts, passing through an intermediate "content" markup, which is provided by Content-MathML for statements, and OMDoc for Views. The idea is that different foundational dialects may be mapped to a same content description, and on the other side, the same content can be translated to different presentational markups.

The choice of using XSLT for performing the transformation (see also [1]) is mainly motivated by the XML framework of the project (MoWGLI was explicitly conceived as a major validation test for XML technology). However, the limited expressive power of XSLT also combines well with a major philosophical commitment of the Helm project, namely that rendering must be a simple operation, not involving major transformations on the proof. The point is that if the rendering operation is too sophisticated we may loose confidence in what has been actually proved in the machine.

3 The C-CoRN repository

The C-CoRN repository [5] consists of a Coq formalisation of basic algebra and basic constructive analysis. The algebra covers an algebraic hierarchy consisting of semi-groups, monoids, groups, rings, fields, ordered fields and metric spaces. Apart from algebra, there is a large amount of analysis, including real numbers, complex numbers, polynomials, real valued functions, differentiation / integration and basic results from analysis like the intermediate value theorem, Taylor's theorem up to the fundamental theorem of calculus and the fundamental theorem of algebra (using both algebra and analysis).

4 Mathematical Notation

Part of the descriptive power of mathematics derives from its ability to represent and manipulate ideas in a complex system of two-dimensional symbolic notations refined during centuries of use and experience.

Especially for didactic reasons it is important to adopt a mathematical notation as close as possible to the usual notation in textbooks. As we will see in Sect. 5.2, this notation is much more dynamic and context dependent than it appears at first sight, posing very interesting and challenging problems not yet solved by the current technology.

A friendly and concise mathematical notation is also required during the formal development of the proofs in Coq. In particular, the syntax and the notational expressivity of Coq have greatly changed in the last releases of the system. However, the priorities are different with respect to a course. In particular, Coq requires notation to be perfectly unambiguous in every situation, severely limiting overloading. It also restricts notation to a mono-dimensional language. As a consequence the notation adopted for the development of C-CoRN and the one that is used in the electronic course will not be the same, preventing an automatic translation in the general case.

4.1 Notation in Helm

The process of associating notation to a formula during rendering is made of two phases. The two phases are both XSLT transformations, since the formula, expressed as a term of the logic of Coq, the Calculus of (Co)Inductive Construction (CIC), is stored in XML. The first transformation is a semantically lossy operation that maps the formula to a MathML Content expression that captures its intended meaning. The second one maps MathML Content to either XHTML or MathML Presentation, only the latter giving access to the most complex bidimensional notations.

The stylesheet to MathML Presentation is quite standard, but for the handling of the layout. Indeed the stylesheet automatically breaks long formulae on multiple lines, exploiting the MathML Content expression to decide where to break the lines and how to indent the expression. Greatly suboptimal with respect to the most natural layout a human can provide, our strategy is superior to the trivial algorithms implemented in the browsers that cannot exploit the content expression. However, layouting greatly increases the complexity of the stylesheet: every time a notation can be applied a template is invoked to estimate the size of the subexpressions and decide if line breaking and consequent indentation are necessary. For instance, detecting and rendering the "less than" relation requires respectively 7, 139 and 142 lines of XSLT for the CIC to MathML Content, MathML Content to XHTML and MathML Content to MathML Presentation transformations.

Providing a new notation, especially in those cases where the notation is quite standard, should not require more than a few seconds, nor any major knowledge from the user, who is not supposed to write about 300 lines of XSLT for each mathematical operator. Since the stylesheets have a very simple and repetitive structure it is possible to automatically generate them from a concise description of the operator, including, for instance, its arity, associativity and type (infix/prefix/postfix) for non binding operators that have a mono-dimensional notation.

To automatically generate stylesheets, Helm provides two intermediate XML languages to describe notation and a set of transformations (in XSLT) which translate the first simplified language in the second, and the second to the three stylesheets CIC to MathML Content, MathML Content to XHTML and

MathML Content to MathML Presentation (see [7]). Moreover, links are automatically added from one level — say, XHTML — to the generating expression at the previous level — MathML Content —, and from the occurrence of the operator (in MathML Presentation or XHTML) to the CIC document where it is defined. Describing the notation for the "less than" operator in the simplified language simply amounts in giving: 1) its arity; 2) its type; 3) its associativity; 4) its URI (unique identifiers) of the operator at the CIC level; 5) the name of the MathML Content element that it must be mapped to; 6) the Unicode symbol that it must be mapped to for presentation. This information can easily be provided by the user by directly editing the XML file. Trusted Logic has also implemented a tool to generate this information from the corresponding one used by Coq to describe notations [6]. The simplified language also recognises binding operators, operators that have arguments that must be left implicit and operators that are rendered as the negation of other operators.

When the simplified language is not expressive enough to describe the notation, the user can directly use the intermediate language, writing the expected representation directly in MathML Presentation (or HTML) extended with special macros to suggest desired positions for line breaking, to insert the required links, to process recursively the sub expressions and so on.

At first the automatic generation of notational stylesheets seems to definitively solve in an elegant way the problem of defining new notations. However, in Sect. 5.2 we will consider the problem of making notations evolve within a document, facing the current limitations of the Helm technology.

4.2 Notation in natural language for predicates

Automatic stylesheet generation has been conceived to translate CIC expressions to their usual mathematical symbolic rendering, for instance to replace "`plus`" by "+". The same technology can be exploited to provide a natural language like notation for predicates. For instance, a generated stylesheet can transform "`(commutes plus nat)`" into "+ is commutative on ℕ", with links from "+" to the definition of "`plus`", from "ℕ" to the definition of "`nat`" and from "is", "commutative" and "on" to the definition of "`commutes`".

Even if the descriptions generated for large formulae are far from being as fluent as those that a human would write, the results are pretty satisfactory and the statements are much easier to understand. The Mizar language has already successfully adopted a similar notation for predicates for years.

From the point of view of the user who must provide the notation, treating words in the same way as symbols is rather onerous. Moreover, the algorithm that automatically breaks a formula into several lines does not produce the expected output for natural language notations, where we would expect long sentences to simply continue on the next line. An improvement of the Helm technology is required to handle this kind of notation in a completely successful way.

5 Documents as views

A course is much more than a set of definitions and theorems. It has its own buildup, which is directed by the rules of didactics and clarity of mathematical exposition. It may sometimes sacrifice formality to provide intuitions in the most direct way, or it may provide the intuitions first and the formal counterpart later on. It contains plenty of examples, exercises and rhetorical text, while omitting technical lemmas and results that will not be necessary or that will be presented only when used.

Thus, to develop our course notes from C-CoRN, it is not a reasonable idea to start from the script, the Coq development where definitions and proofs are given in their order of definition, and integrate it with examples and the rest. On the contrary we decided to start from the buildup of the course, seen as a huge hypertextual electronic "document" (divided into chapters, sections and the like) in which we embed formal definitions, lemmas, exercises and references to definitions and lemmas in the middle of informal sentences.

The embedding does not need to be done statically. On the contrary we may imagine a document — called *view* in the Helm terminology — where special elements are put where formal notions need to be embedded. The view can be written in any XML language we like, and XML namespaces are used to avoid confusion with the special elements. To visualise a view, the document is processed on the fly by the Helm processor and mapped to a standard XHTML page (eventually embedding MathML Presentation islands and made dynamic by means of JavaScript). All the special elements are expanded by the processor which embeds in the document the requested rendering of the formal notions.

To speed up the development time and to be sure of the didactic quality of the course obtained we decided to exploit the course organisation, the rhetorical text and all the exercises and examples (the latter to be first formalised in C-CoRN that used to lack examples completely) from IDA.

IDA [4] is an interactive course on algebra, which has been developed at the Eindhoven University of Technology (NL), for first year mathematics students. The IDA course notes have been developed over the years in the context of a first year class on algebra. It consists of a book with a CD-ROM. (Before the book + CD-ROM were published by Springer, β-versions of the material were available through the web.) The material on the CD-ROM is the same as in the book, but there is additional functionality:

- Alternative ways of browsing through the book.
- Hyperlinks to definitions and lemmas. However, since the hyperlinks have been inserted manually, many of them are missing, in particular in the textual flow.
- Applets that show algorithms (e.g. Euclid's algorithm).
- Multiple choice exercises.

IDA does not provide a formal treatment of proofs. Proofs in IDA are like proofs in ordinary mathematical text books: plain text, so we couldn't reuse any proof from IDA. However it is very instructive to compare the proofs in IDA,

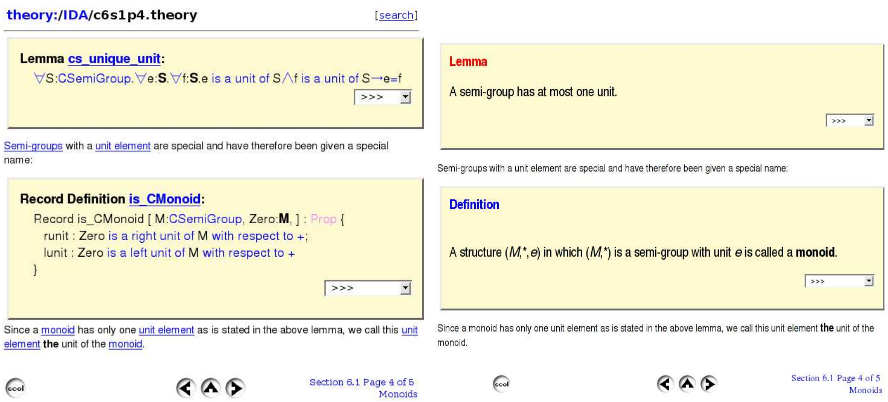

Fig. 2. Comparison between our course notes and IDA.

targeted at an audience of students (and developed in classical mathematics), with the formalised (constructive) ones in C-CoRN. Because the structures in IDA, like monoids and groups, are not as rich as the ones in C-CoRN, which include also an apartness relation, the C-CoRN proofs are in general a bit more involved than the original IDA ones. However, we have only come across one theorem that was false constructively.

Concretely, our goal was to obtain electronic course notes from the C-CoRN repository having the same look and feel as IDA, using Helm-tools to render mathematical objects (definitions, statements and their proofs) and to create hyperlinks between them.

Fig. 2 compares the page that defines a monoid in our course notes (on the left hand side) with the corresponding page in IDA (on the right hand side). The two pages are similar and the interactivity of the IDA page has been preserved: the lower frame is used to navigate in the notes and the drop down boxes give access to examples and exercises which are the same in the two courses. Our page is richer in functionality. First of all it adds another navigation frame on top which provides a breadcrumb trail and a link to the Whelp search engine. Whelp [3] is a search engine developed in MoWGLI to index and retrieve proofs and definitions in a formal library by means of powerful queries over the mathematical expressions in the statements. As IDA does not have any search or index facilities, this is a pure gain from the formalisation. We also observe that our course offers many more links to definitions than IDA. Indeed every occurrence of a formal concept is given its own link, both in the formal statements and definitions that are automatically generated from C-CoRN and in the free text that comes from IDA.

Finally we notice remarkable differences between C-CoRN and IDA for the statement of uniqueness of a unit and the definition of a monoid. One of the reasons for the increased verbosity of the formal definitions is that the formal library lacks or does not exploit a few notions that help to make the statements more concise. For instance, it would be possible to define the notion of uniqueness over a type T and a property over T, using it to state the lemma of uniqueness of the unit element. This is not normally done when working in Coq since Coq unification does not automatically expand this kind of general notions, making proofs much more cumbersome for the user. The same phenomenon appears in the definition of a monoid. Instead of relying on the notion of unit, in C-CoRN it is preferred to explicitly state that `Zero` is both a left and a right unit, since in Coq the constituents of a conjunction are not automatically derived from the conjunction when appropriate.

If the general notions were used, we believe that the differences between the formal and informal versions would be less relevant and a bit of extra notation would provide a rendering that is not more cumbersome than the human provided counterpart (even if the latter would remain much more natural).

5.1 In-line rendering

A typical mathematical document will not only contain raw, precise, rigorous mathematics, but also sentences whose meaning is more fuzzy or non-mathematical, somewhere in the range between normal English text and rigorous mathematics. This "free text" can contain for example historical remarks or remarks such as (this is an actual excerpt from IDA)

> Most binary operations in which we are interested distinguish themselves from arbitrary ones in that they have the following property.

The meaning of "in which we are interested" cannot be expressed in a theorem prover and escapes its checking framework, but that kind of text is still an important part of a mathematical document. However, it still speaks about and refers to formally defined mathematical notions ("binary operations"). It is thus desirable to be able to have free-form (non proof assistant checked) text that nevertheless uses names and notations of the proof assistant checked "library of mathematics" the document is about.

The initial implementation of the Helm system only allowed to refer to a mathematical notion by embedding its whole definition, as a separate paragraph, right there in the document. To complete our task we have implemented the possibility to have a rendering that

- integrates in a sentence, as a word or phrase, instead of being its own paragraph. We call this the *in-line* rendering.
- removes (or adds) some parts of the definition of the object that are included (or not included) by default, such as the name of the object, the body of the definition or the notation for the object.

These choices must be made by specifying attributes to the XML element that is used to provide a link to the formal notion and to ask for the embedding of its rendering in place of the element. Even if in principle all these choices are orthogonal, not every combination makes sense and gives a reasonable result.

To represent the free text of IDA in our course notes we have also sometimes explicitly chosen to avoid references to formal objects even if they were available. This was done in cases where some particular translation of the formal expression was necessary for the general flow of the document, to rise from the level of "sequence of mathematical statements" to the level of "document that tells a story". For example, the very first sentence of the chapter we have treated:

> The map that takes an element of $\mathbb{Z}$ to its negative is a unary operation on $\mathbb{Z}$, while addition and multiplication are binary operations on $\mathbb{Z}$ in the following sense.

Mathematically, this says the same as "$-_{\mathbb{Z}}$ is a unary operation on $\mathbb{Z}$ and $+_{\mathbb{Z}}$ is a binary operation on $\mathbb{Z}$ and $*_{\mathbb{Z}}$ is a binary operation on $\mathbb{Z}$" (which is approximately how Helm would have rendered the corresponding formal mathematical statement). For a human reader, however, "while" does not just express a conjunction, as far as quality (e.g. ease of reading, clarity of point) of the document is concerned, even though "while" and "and" are equivalent mathematically. A document making absolutely no use of this kind of subtleties of language would be quite "dry" and hard to read for a human. Also notice the contraction of "$+_{\mathbb{Z}}$ is a binary operation on $\mathbb{Z}$ and $*_{\mathbb{Z}}$ is a binary operation on $\mathbb{Z}$" into "$+_{\mathbb{Z}}$ and $*_{\mathbb{Z}}$ are binary operations on $\mathbb{Z}$".

5.2 Context depending rendering

In Sect. 4 we have described the Helm facilities to provide mathematical notation easily, and we claimed that at first sight— when considering the rendering of a single definition or theorem at a time in a sort of vacuum — the machinery seems to be expressive enough. As soon as we started to consider the rendering of views, however, we realised that mathematical notation is much more dynamic and context dependent than what it seems to be at first sight. We will examine a few examples where the current context-free machinery of Helm is not sufficient.

Notations depending on lemmas Stating a lemma can implicitly change the notation used in the rest of the view. The most well known example is proving the associativity of a binary operation. Until associativity is proved, every expression involving the operator must be fully parenthesised. However, once associativity is known, parentheses are omitted. Since "most of the time" associativity is known (i.e. the majority of the theorems depend logically on the lemma of associativity), we could have decided to always omit parentheses. However, this solution is not satisfactory since we sometimes face statements that in this way are reduced to trivialities. For instance, the lemma of associativity for $\circ$ would become $\forall x, y, z. x \circ y \circ z = x \circ y \circ z$.

Another similar annoying example can be found in the theory of semi-groups where first the notion of *being a unit* is defined and only after that the uniqueness of the unit in semi-groups is proved. So when we state the uniqueness of the unit, we would like to speak about "*a* unit", because the uniqueness has not yet been established:

Lemma 1. *∀S:CSemiGroup.∀e:S.∀f:S. e is a unit of S∧f is a unit of S→e=f*

But after we have proved the uniqueness, we would like to see "*the* unit" in all following uses.

It has also been argued that this is not really a problem, because differences in the English language — like "a" and "the" here — reflect also a difference in the mathematical meaning: "a unit" would denote a relation and "the unit" would denote a function. So, it has been proposed to use "a" whenever we see the relation, and to use "the" whenever we see the function. However, it is not clear that the correspondence between the English language and the mathematical meaning is as straightforward as suggested here. It is not a mistake to use the relation even after uniqueness has been established, but to use the indefinite article in English while we know the uniqueness, seems highly unusual.

Overloaded notations It is pretty common in mathematics to overload notations. However, sometimes the two semantics attached to a notation are needed at the same time and the context may not be sufficient to disambiguate. Then, the notation for at least one of the semantics must be changed, for example by further qualification. In Helm we can easily overload a notation and we can also provide qualified notations. However, the choice of using a qualified notation depends heavily on the context and cannot be made automatically.

Multiple notations for (instances of) abstract notions When defining an abstract notion, say a group, a default notation is also usually defined to state the general theorems about the abstract notion. For instance, we could choose a multiplicative notation as the default for an abstract group. Later on abstract notions are instantiated to concrete ones and different notations should be applied to the different instances. For instance, for particular groups we often prefer an additive notation.

In the formalisation, a group becomes a record (a tuple with named projections) whose first element is the carrier and whose second element is the operation on the carrier. Thus an occurrence of the operation of a group is formally represented as the projection π_2 applied to the group G: $(\pi_2\ G)$. The operation applied to two arguments x and y becomes $(\pi_2\ G\ x\ y)$. In Helm notations must be associated to constants and are independent of the arguments the constants are applied to. In this case we can associate a notation to π_2 — say the additive notation — by saying that π_2 is a binary infix operator with an implicit argument. Thus $(\pi_2\ G\ x\ y)$ is represented as $x + y$. We could make the implicit argument explicit, representing the previous expression as $x +_G y$ equally easy (notice, however, that G can be a huge expression). However, we have no way of saying that $+$ must be replaced by $*$ for some particular groups G.

Having seen a few examples where the notation depends on the context, we will now discuss a possible enhancement of Helm to allow dynamic notation.

5.3 Extending Helm with context dependent notation

As we have seen in the previous sections, there are several different kinds of dependency on the context. Here we discuss the ways these could possibly be dealt with in Helm.

The first one is the case of temporary dependencies on lemmas and definitions, where a lemma or definition that occurs *in the view*, i.e. in a single page in our electronic notes, activates or changes a notation. Since the order of appearance in a view is different from "logical causality" (i.e. the partial order induced when a lemma refers to previously defined lemmas and definitions), we need to search for a solution by associating extra information to a view. In particular we may apply a batch process to a view that collects for each formal notion referred in the view all the formal notions that occur before it. Notice that a view can follow other views in the intentions of the authors, i.e. all the pages that precede it. Thus the batch process should visit all the views in their order of dependencies to collect the information to associate to objects. Once the information has been collected, we can identify for each object the notations that are active when presenting it and store this as metadata in the Helm database, associating it to the occurrence of an object in a particular view. The notational stylesheets can retrieve from the database the metadata (in RDF format) and apply only the templates supposed to be active.

Even if the previous solution seems satisfactory at first, it induces a new problem, e.g. when the user follows a hyperlink from an occurrence of a notion in a view to the definition of the notion. This definition exists in a vacuum with respect to the view that contains the hyperlink: a notion is not required to be defined in the views it is used in and it can be used in multiple views unrelated to the current one. This precludes any contextual metadata from being used in the decision of which notation to apply. Theoretically, the solution consists in considering the vacuum as a new view generated on the fly that inherits its metadata from the view the user is coming from. Practically, the Helm tools cannot handle this operation right now and implementing the new functionality requires a significant overhaul of the design of the tools.

The second case of contextual dependency, overloading, can be handled in a similar way. Indeed we can just give the user the possibility to associate to each lemma or definition metadata that says that some overloaded notation needs to be qualified. The main difference with the previous case is that the metadata is likely to be associated to the object in any possible view, and not to the occurrence of the object in a particular view.

The third case is the dependency of the notation on the arguments of the operator, which is much harder. Currently we have no practical solution. The problem becomes particularly complex when the user follows a hyperlink to the proof of a general statement. Since the statement was applied to an argument in the current view, its notation is supposed to depend on the argument. However, in the new window that shows the statement the actual arguments are universally quantified and the notation is likely to change accordingly to a default value (e.g. from additive to multiplicative). While this behaviour can be confusing, the situation on this point is exactly the same as with classical paper documents: The reference text book about a notion may use a notation that is different from a particular instance of the notion in another document. One would hope, however, that semantically rich electronic documents would improve the situation here.

6 Proofs

6.1 What is a formal proof?

Rendering of formal proofs is a complex task. The first issue that arises when rendering formal proofs is: what kind of "object" is a formalised proof? This very much depends on the proof assistant used. The user interacts with a proof assistant via a so called *proof script*, the sequence of input commands that a user enters to prove a result. An important difference between proof assistants lies in the style of the language used in these scripts. Roughly speaking there exists two styles: *procedural* and *declarative*. In the procedural style (Coq, NuPRL, HOL, Isabelle), a user inputs *tactics*, commands that *modify the state of the system* (e.g. by applying a logical rule or a hypothesis or calling an automation procedure). So in a procedural proof style the user tells the system what to *do*. In the declarative style (Mizar, Isabelle/Isar) a proof script is essentially a sequence of intermediate results that *tell the system what our knowledge state is*. These go together with hints (known lemmas) on why the alleged intermediate result should hold. In a declarative proof style, the user tells the system where we *are*, and the built-in automation should verify that that's indeed the case.

In procedural proof assistants there may be another notion of proof, which is a *proof object* or proof term. This is a mathematical object (typically a typed lambda term) composed of only the very primitive logical rules. A tactic script creates a proof object, which is usually quite big, but has the advantage that it is directly verifiable by a small and trusted kernel. The Coq system that we use in C-CoRN has these proof objects.[5]

6.2 What proof do we store and what proof do we render?

The problem of procedural scripts is that the syntax and semantics of tactics is very system dependent and changes rapidly together with the overall system evolution. Declarative scripts are more robust, but similarly changes in the automation engine of the system can break a declarative script. As a consequence, scripts cannot be reasonably used for long term preservation of the information, and are also hardly reusable by any external, third party tool. Proof objects have a very clean and formal semantics, defined by the underlying foundational system (that is usually quite stable, even along the evolution of the tool). The main drawback of proof objects is that they are quite verbose and could be far

[5] In systems like HOL and Isabelle no proof object are created, but in principle one could let the tactics create a proof object on the fly. In Mizar, the declarative proof itself is seen as *the* proof

away from the original proof of the user (which is usually better reflected by the tactic macrosteps of the proof script).

The Helm Project is particularly focused on the long term preservation and management of repositories of formal knowledge. To this aim, as explained above, the most relevant information is provided by proof objects. In the framework of the MoWGLI project we have developed an exportation module able to extract from the Coq proof assistant the raw lambda terms encoded into a suitable XML dialect[6]. Helm also provides stylesheets attempting a natural language reconstruction of the proof object; since the lambda term is isomorphic to a natural deduction proof, the result is, if not really appealing, surely readable.

In line with to the philosophy described in section 2, we make no major transformation on the input lambda term (the proof object) before presenting it. It is obvious that with a little effort we could easily improve the presentation by suppressing (or removing) a lot of detail. A typical example are proof objects that are found by an automatic decision procedure: these are usually extremely verbose and complex proofs that humans don't want to read. Not showing these subproofs' details by default would probably work well. However, in the proof object as such we can't trace the application of an automation tactic, so this would require a major transformation of the term before rendering it. A particularly fortunate case of an automation tactic in Coq is that of a *reflexive tactic*. This is not the place to go into detail about the nature of reflexive tactics, but the crucial point is that it does not create a huge proof term. Instead it encodes the goal to be proved as a syntactic object and then applies a generic lemma to an argument that solves the goal by mere computation. (And in Coq, computation does require a proof.) So, to detect a reflexive tactic it is sufficient to recognise (in the stylesheets) an application of the generic lemma and hide its rendering.

6.3 The actual rendering of the proofs

Thanks to the Helm exportation module we have produced proof terms for each C-CoRN theorem and we have use the standard Helm technology to render in our course the natural language generated from them. This uses the standard well-known transformation of typed lambda terms to natural deduction proofs. If rendering the proof term has not required major changes in the Helm technology, on the contrary we had to sensibly augment our library to match the textual flow of IDA. Below we give an example of a proof in C-CoRN, as rendered by the Helm tools. We observe that the C-CoRN proof is much more verbose than the one from IDA, which just reads

$$e = e + f = f.$$

[6] We have also attempted to define a similar exportation functionality for Coq*proof trees*, which integrate information from the proof script with the proof objects, but we eventually failed. This failure was due to the complexity of the data structures (e.g. handling of bound variables) and the fact that the proof trees are much more subject to changes from one version of the system to the next.

On the other side, the C-CoRN proof gives the full formal details. Some of the details are "hidden" under a green link: clicking this link unfolds the details on request. In the black-and-white printout: these links are the crossed box before "Assumptions", which unfolds explicitly the local assumptions of the lemma, and the "Proof of $e = e + f$", which gives an explicit proof of this fact.

```
cic:/CoRN/algebra/CSemiGroups/cs_unique_unit.con     [search]

DEFINITION cs_unique_unit()
TYPE =
      ∀S:CSemiGroup.∀e:S.∀f:S.e is a unit of S∧f is a unit of S→e=f
BODY =
      ⊞Assumptions
        we must prove e is a unit of S∧f is a unit of S→e=f
        or equivalently (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)→e=f
        suppose H: (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)
          consider H
          we have:
          (H0) ∀a:S.e+a=a∧a+e=a
          (H1) ∀a:S.f+a=a∧a+f=a
          by (H0 .)
          we have:
          (H2) e+f=f
          (H3) f+e=f
          by (H1 .)
          we have:
          (H4) f+e=e
          (H5) e+f=e
          we have the following chain of (in-)equalities:
            e
               by
                Proof of  e=e+f
             = e+f
               by  H2
             = f
          we proved e=f
        we proved (∀a:S.e+a=a∧a+e=a)∧(∀a:S.f+a=a∧a+f=a)→e=f
        that is equivalent to e is a unit of S∧f is a unit of S→e=f
      we proved ∀S:CSemiGroup.∀e:S.∀f:S.e is a unit of S∧f is a unit of S→e=f
```

7 Conclusions

This research has been a test case for the use of a formal library, especially C-CoRN, in a mathematical document. It has also been a test case for the Helm tools in supporting the creation of such a document. This has led to interesting refinements of the Helm tools, as described in Section 5.

The use of the mathematics in the formal library has some huge advantages: one can use everything that is already in the library. In our case, not much from what we needed was present in C-CoRN when we started: only about 20% of the definitions, lemmas and examples we have used in formal form in our final document. Considering that the document covered about ten pages of basic mathematics and that we have not replaced every definition, lemma or example, that seems a bit disappointing. But it has been very useful to us anyway, because many items that were missing in the library were fairly easy to formalise from lemmas that were present. And by adding items to the library, we have made a contribution to future use. So, in short, it is beneficial to use a library because it is *economical*: if something has been defined or proved, it can be used over and over again. Nothing has to be done more than once.

It should also be noted that, using a formal library within an existing mathematical document (IDA) by putting references to it does note guarantee the *coherence* of the document. A mathematical document aims at satisfying a strict requirement: objects and lemmas are not to be used before they have been defined or proved. Of course, sometimes a lemma is used and is only proved later at a more suitable place, but this is mostly announced in the surrounding text and this can be seen as the use of a local assumption, instead of premature use of a lemma. But using an object or a lemma that is unfamiliar to the reader without any explanation, can be seen as a mistake. Because we use an existing library (C-CoRN) and in the (IDA) document we just put references to the library, we have paid no attention to this kind of logical coherence in the IDA document. It could well be that the order of the references does not meet the logical order of the corresponding items in the library. The fear for logical mistakes *between* items, as opposed to logical mistakes *within* items (proofs and definitions), is not imaginary. In the ten pages of IDA that we have looked at, the logical coherence has been violated several times. A positive aspect of our development is that the Helm-links from IDA provide the possibility to track the formal dependencies (inside C-CoRN) of the statements that we find in IDA.

The order in which the Helm system forces the user to make the document, i.e. first formalise the mathematics and only after that describe its rendering can sometimes be inconvenient. In Helm the link from the informal description to the formal object is made by the user, who writes (in the XML description) explicitly where the object can be found. This means that whenever an object in the library moves, the user has to alter its XML description. And objects do move, because the library is not a static object: theory files are reorganised and are being split up when they become very large. One may expect these libraries to stabilise at a certain point, but C-CoRN is still very much a dynamic library.

Acknowledgements We thank the anonymous referees for their useful comments. We have not yet been able to treat them all, but we will try do so in the final version.

References

1. A.Asperti, L.Padovani, C.Sacerdoti Coen, I.Schena. *XML, Stylesheets and the re-mathematization of Formal Content*. Proceedings of "Extreme Markup Languages 2001 Conference", August 12-17, 2001, Montreal, Canada.
2. A. Asperti, L. Padovani, C. Sacerdoti Coen, F. Guidi, I. Schena. Mathematical Knowledge Management in HELM. Annals of Mathematics and Artificial Intelligence, 38(1): 27–46; May 2003.
3. A. Asperti, F. Guidi, C. Sacerdoti Coen, E. Tassi, S. Zacchiroli. A content based mathematical search engine: whelp. Submitted for publication.
4. A. Cohen, H. Cuypers, H. Sterk, *Algebra Interactive!*, Springer 1999.
5. L. Cruz-Filipe, H. Geuvers and F. Wiedijk, C-CoRN, the Constructive Coq Repository at Nijmegen. In: A. Asperti, G. Bancerek, A. Trybulec (eds.), *Proceedings of MKM 2004*, Springer LNCS 3119, 88-103, 2004.
6. E. Gimenez, *Validation 2: Smart Card Security*, MoWGLI deliverable, 2005.
7. P. Di Lena, *Generazione automatica di stylesheet per notazione matematica*, Master's thesis, University of Bologna, 2002.