INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Report n. D6.a
An Interactive Algebra Course with Formalised Proofs and Definitions

Andrea Asperti, Herman Geuvers, Iris Loeb, Lionel Elie Mamane, Claudio Sacerdoti-Coen

Project Acronym: MoWGLI
Project full title: Mathematics On the Web: Get it by Logic and Interfaces
Proposal/Contract no.: IST-2001-33562 MoWGLI

Contents

1 Introduction 3

2 Formalising the mathematics 4
2.1 Handling non-constructive proofs . . . 4
2.2 Stripping Type valued universal quantifications of lemmas . . . 6

3 Helm enhancements during IDA-in-Helm 7
3.1 Links . . . 7
3.2 Coercions handling . . . 8
3.2.1 The `d_c` stylesheet . . . 8
3.2.2 IDA-in-Helm enhancements . . . 8
3.3 In-line Rendering . . . 9
3.4 Automatic generation of the notational stylesheets . . . 10
3.4.1 Introduction . . . 10
3.4.2 First implementation for automatic generation . . . 10
3.4.3 Changes . . . 11

4 Discussion 17
4.1 Independent products . . . 17
4.2 Context dependent rendering . . . 17
4.3 Moving mathematical objects . . . 17
4.4 Using reflection . . . 18
4.5 Use of a formal library: economy vs. coherence . . . 18

5 Conclusions 19
5.1 Analysis . . . 19
5.2 Future Outlooks and remaining problems . . . 19
5.3 Conclusion . . . 20

1 Introduction

One of the aims of the Mowgli project is to develop the functionality to generate web-based interactive mathematical documents out of a repository of formal mathematics. One of the test cases that was defined for Mowgli was the applicability for course notes. The present report describes and discusses this test case.

The first issue was to discuss what exactly to develop. In Mowgli, we had already developed the Helm functionality, allowing to extract an XML encoding out of a piece of mathematics that is formalised in the proof assistant Coq. Furthermore, we had already developed the functionality to generate semantical and presentational mark-up (using style sheets) for such an XML encoding of formalised mathematics. Finally, we had a large repository of Coq-formalised constructive mathematics, C-CoRN [4], existing of basic algebra and analysis.

The – maybe most obvious – idea of just using the Helm functionality to extract web-based documents out of the C-CoRN repository didn't seem to be the right way to proceed: a set of course notes has its own build-up, which is directed by the rules of pedagogics and clarity of exposition and which may be orthogonal to the order that is derived from the formal repository. We therefore decided that we would start from an existing set of course notes. For this we took IDA [2], an interactive course on algebra, which has been developed at Eindhoven University of Technology (NL), for first year math. students. The course notes consist of a book with a CD-rom. (Before the book + CD-rom were published by Springer, beta-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 like

- Alternative ways of browsing through the book,
- Hyperlinks to definitions and lemmas,
- Applets that show algorithms (e.g. Euclid's algorithm),
- Multiple choice exercises.

As the IDA course notes have been developed over the years in the context of a first year class on algebra, we felt that this would provide the right set up for our test case. What IDA does not provide, and something that the Mowgli tools should be able to add, is a formal treatment of proofs. Proofs in IDA are like proofs in ordinary math. text books: plain text. Something else that the Mowgli tools should be able to add is a formal treatment of definitions and lemmas, which should become formal objects that can be rendered as ordinary mathematics. Moreover, the hyperlinks (in IDA) to definitions and lemmas should now become hyperlinks to these formal mathematical objects.

So our concrete goal was to use the Mowgli tools to generate formal content in IDA in the following ways.

1. Let the mathematical statements (definitions and lemmas) be rendered by the Mowgli tools out of the formal statements in C-CoRN,
2. Let the proofs in IDA be rendered by the Mowgli tools out of the formal proofs in C-CoRN,
3. Let the examples in IDA be rendered by the Mowgli tools out of the formal definitions and lemmas in C-CoRN,
4. Let the hyperlinks (to definitions and lemmas) in IDA be rendered by the Mowgli tools out of the formal links in C-CoRN.

So we certainly do not expect the Mowgli tools to be able to generate a set of course notes out of a repository of formal maths, but we do expect them to be able to generate the necessary links and to do the proper rendering.

How to evaluate the Mowgli tools on the basis of this experiment? The following issues are relevant to look at.

- Are the above possible at all.
- What is the quality of the rendered statements, compared to the original statements in IDA. (Mathematicians have various, often implicit, notational standards that they use very flexibly. Can we mimic that?)
- What is the quality of the rendered proofs, compared to the original proofs in IDA. (Mathematical proofs have a lot of hidden details that our tools should be able to show on demand. Can we do that?)
- How user friendly is this. (What should an author, say someone that is writing the new version of IDA, do to achieve all this?)

In this report we will discuss the technical aspects of our work and then we will draw some conclusions. An important part of the work was to produce the style sheets that do the rendering. That is described in Section 3.4. As we wanted to connect IDA to the formal content of C-CoRN, we had to make sure that the material of IDA was covered by C-CoRN. This was not the case, so we had to extend C-CoRN. Most notably, mathematical course notes contain many examples, e.g. after the notion of "semi-group" is introduced, it is shown – as an example – that $(\mathbb{N}, +)$ forms a semi-group. In C-CoRN, this amounts to introducing an explicit definition of the structure $(\mathbb{N}, +)$ of type "semi-group" (which involves proving that $+$ is associative etc.) Also the constructive aspect of Coq (and C-CoRN) involved a bit of extra work, as some of the statements in IDA are not constructively provable, so we had to change them a bit. In Section 3 we will discuss the technical aspects of the work and in Section 5 we will discuss the whole work: what are the pros and cons of using a formal mathematical library for a set of course notes like IDA and what are the conclusion with respect to the points raised above.

2 Formalising the mathematics

2.1 Handling non-constructive proofs

C-CoRN ([1]) is a library of constructive mathematics, and IDA ([2]) is a course in classical algebra, although it has a special focus on algorithms. Because the structures in C-CoRN are all equipped with an apartness relation, it meant that we had to prove more than has been proved in IDA. Instead of showing, for instance, that some set with an operator and an equality was a semi-group, we had to define also an apartness relation on this set and show that it was a constructive semi-group. So we had to do a bit more work, but in general it did not trouble us much.

Only in one place the disagreement of IDA and C-CoRN concerning the logic was problematic: the characterisation of cyclic monoids. First for every $k, n \in \mathbb{N}$, $n > 0$ a cyclic monoid $C_{k,n}$ is defined. The carrier of this monoid is the set $\{e, c, \dots, c^{k+n-1}\}$ and the operation is defined as follows:

- $c^j * c^i = c^{j+i}$, if $j + i < k + n$;
- $c^j * c^i = c^{k+((j+i-k)\mathrm{mod}n)}$, if $j + i > k + n - 1$;
- c^0 is the unit e.

Figure 2.1 shows a visualisation of the orbit of e under $*c$.

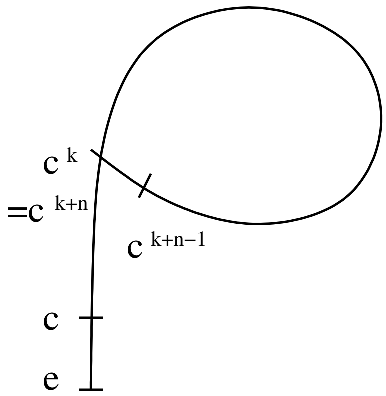


Figure 1: The orbit of e

Theorem 2.1 (Characterization of cyclic monoids.) *Every cyclic monoid is isomorphic with either $C_{k,l}$ for certain $k, l \in \mathbb{N}$ or $(\mathbb{N}, +, 0)$.*

We will show with a Brouwerian counter-example, that this is not true constructively.

Counterexample 1 *Let k_{99} be the escape number of π. Define $M := \{n|n \in \mathbb{N}|n < k_{99}\}$. Define the operation $*_M$ by:*

$$i *_M j = i + j \ mod\ k_{99}.$$

Then it is clear that $(M, *_M, 0)$ is a monoid. But saying that $(M, *_M, 0)$ is isomorphic with either $C_{k,l}$ for certain $k, l \in \mathbb{N}$ or $(\mathbb{N}, +, 0)$ implies that the decimal expansion of π contains a block of 99 nines or it doesn't, which is audacious.

Because we could not thus have any hope to prove this theorem in Coq in the form it was presented in IDA, we have adapted the theorem such that it could be proved therein. To understand our adaptation, it is explanatory to study the structure of the original proof.

Proof Suppose that $(C, *, e)$ is a cyclic monoid generated by the element c of C. We make the following case distinction.

- **There are $k < l$ with $c^k = c^l$**
  Let k and l be the smallest pair (in lexicographical order) with this property.
  ⋮
  ⟨construction of a map from $C_{k,n}$ for certain $k, n \in \mathbb{N}$ to C and a proof that it is an isomorphism⟩
  ⋮
  Hence C is isomorphic with $C_{k,n}$ for certain $k, n \in \mathbb{N}$.

- **There are no such k and l.**
  ⋮
  ⟨construction of a map from $\mathbb{N}$ to C and proof that it is an isomorphism⟩
  ⋮
  Hence C is isomorphic to $(\mathbb{N}, +, 0)$.

Maybe the first thought that comes in mind is to require the decidability of the equivalence relation. But this is not enough to justify the case distinction (see Counterexample 1). Therefore we have split the theorem in two and we have considered the parts separately:

- Every cyclic monoid generated by an element c such that there are $k < l$ with $c^k = c^l$ is isomorphic to $C_{n,m}$ for certain $n, m \in \mathbb{N}$.
- Every cyclic monoid generated by an element c such that there are no $k < l$ with $c^k = c^l$ is isomorphic to $(\mathbb{N}, +, 0)$.

To prove the first part, it seems enough at first sight to require the decidability of the equivalence relation, because when we know there are $k < l$ with $c^k = c^l$, it enables us to conclude there is a smallest pair with that property, i.e.

$$\text{there are } k < l \text{ such that } c^k = c^l \text{ and for any } k_1 < l_1\text{: if } (k_1, l_1) < (k, l)\text{, then } c^{k_1} \neq c^{l_1}.$$

However, because we have to prove strong version of injectivity of the function constructed in the proof, we would like to have a stronger assumption at our disposal:

$$\text{there are } k < l \text{ such that } c^k = c^l \text{ and for any } k_1 < l_1\text{: if } (k_1, l_1) < (k, l)\text{, then } c^{k_1} \# c^{l_1},$$

or equivalently

$$\text{there are } k < l \text{ such that } c^k = c^l \text{ and for any } k_1 \neq l_1\text{: if } (k_1, l_1) < (k, l)\text{, then } c^{k_1} \# c^{l_1}.$$

This seems provable from the decidability of the apartness relation. To make it easy on ourselves, we have not assumed the decidability of the apartness relation and proved this property, but we have assumed this property immediately.

The proof of the second part is now straightforward. So the theorem we have proved eventually in Coq is:

Theorem 2.2

Part I *Every cyclic monoid generated by an element c such that there are $k < l$ with $c^k = c^l$ and for all $k_1 \neq l_1$: if $(k_1, l_1) < (k, l)$, then $c^{k_1} \# c^{l_1}$, is isomorphic to $C_{n,m}$ for certain $n, m \in \mathbb{N}$.*

Part II *Every cyclic monoid generated by an element c such that for all $k < l$: $c^k \# c^l$, is isomorphic to $(\mathbb{N}, +, 0)$.*

2.2 Stripping Type valued universal quantifications of lemmas

As discussed in [3], for computational reasons the C-CoRN library has two types for propositions: `Prop` and `CProp`. At first `CProp` was a synonym for `Set`, but later, when for good reasons the type of `CSetoid` became `Type`, `CProp` had to change to `Type` too. We work in this way with synonyms to distinguish data-types (in `Set`, `Type`) from propositions (in `Prop`, `CProp`). This difference is crucial for a good rendering.

Right from the start the rendering of propositions in HELM gave problems. Because the XML exportation unfolded the type `CProp`, many propositions were presented like definitions. But even when the exportation module had been improved such that explicit casts to `CProp` were reflected in the XML, some of the propositions were still printed wrongly. The cause of these errors appeared not to lie in the exportation module or in any part of HELM. The problematic propositions appeared simply not to have type `CProp`, but to have necessarily type `Type` instead.

This is the Π-rule for types in Coq:

$$\frac{\Gamma \vdash A : \mathtt{Type}_i \quad i \leq k \quad \Gamma, x : A \vdash B : \mathtt{Type}_j \quad j \leq k}{\Gamma \vdash \Pi x : A.B : \mathtt{Type}_k}$$

Now, `CProp` is `Type`. This means that `CProp` is $\texttt{Type}_j$ for a fixed – but to us unknown – number j. The carrier of a `CSetoid` has type `Type`. But opposed to `CProp`, this `Type` is not fixed. We allow the carrier of a `CSetoid` to have type $\texttt{Type}_i$ for *any* number i. A statement like

```
Variable S:CSetoid.

Lemma ap:  forall (x y:S), x[#]y.
```

can be understood as follows. Take an arbitrary `CSetoid` S. The carrier of S is of type $\texttt{Type}_i$ for a certain number i. The sort of `ap` is now $\texttt{Type}_k$ for a certain number k. We can not force the sort of `ap` to be `CProp`, or in other words: we can not force k to be j, because we have taken an *arbitrary* `CSetoid`. Once this lemma has been proved and the section has been closed, we can instantiate S with any `CSetoid`, also with ones of type $\texttt{Type}_i$ where $i > j$. So the idea to have all data-types in type `Type` or `Set` and all propositions in type `Prop` or `CProp`, can not be realised with the current definitions of C-CoRN.

To see our lemmas as lemmas and not as definitions in HELM, we have applied a "trick" in the Coq file. Whenever we see a universally quantified computationally relevant proposition, we have put this lemma in a section and we have stripped of all quantifications and made them into section variables until the remaining lemma had type `CProp`. So for lemma `ap` this would be:

```
Variable S: CSetoid.
Variable x y:  S.

Lemma ap:  x[#]y.
```

To come to a more satisfactory solution, it has been proposed to make the type universes in C-CoRN explicit, in such a way that the types of data-types are all lower than `CProp`. This has yet to be explored.

3 Helm enhancements during IDA-in-Helm

3.1 Links

As hypertext documents, Helm documents contain links to other (Helm or non-Helm) documents. The architecture of the system, encoding the whole state of the user session in the URL, means that author-level URIs (the addresses the author of the document writes in the links of his document), like `cic:/CoRN/algebra/foo` have to be dynamically rewritten to system-internal URLs (the address the user's browser sees, an uwobo request), like `http://mowgli:5081/apply?keys=...`

Before this "validation for education documents" effort, such links could only be pointing to other Helm documents, they had to be absolute links and these links had to be "normal" XHTML links (`A` tags with an `HREF` attribute): the machinery was rewriting *all* and only `A`/`HREF` tags. IDA suffered from both these limitations: First, it contains document parts not be helmised, such as images, javascript code and java code, that are included in the document by reference. Second, its intra-document linking is done through a pull-down menu and not "plain" XHTML links.

The helm URI-to-URL rewriting engine thus needed two extensions:

- The ability to have (relative or absolute) links to non-Helm resources. These relative links have to be relative to where the "raw" Helm document is (where the getter gets it from) and not relative to the URL seen by the browser, namely an uwobo service request. They thus have to be rewritten to absolute URLs for the browser.
- The ability to have links to Helm documents more or less anywhere, not only in `HREF` attributes.

This has led us to redesign the URI-to-URL rewriting engine to act only on an explicit request, in form of a `helm:helm_link` or `helm:external_link` attribute to any node; the value of this attribute gives the name of the attribute of the same node that needs to be rewritten. In this way, any attribute of any XML node can undergo URI-to-URL rewriting. The getter now puts an `xml:base` tag in the document giving where it got it from, and the rewriting engine uses that information to unrelativise relative links.

3.2 Coercions handling

3.2.1 The `d_c` stylesheet

The `d_c` (for "drop coercions") stylesheet is a preliminary step to the CICML $\longrightarrow$ MathML Content transformation that gets the rest of the chain rid of a Coq technicality, namely *coercions*.

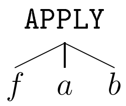
Coercions are essentially a way to get around the lack of subtyping in the type theory behind Coq, the CIC. They are essentially "type casts", a function that maps a of type A to its canonical counterpart in type B. For example, it extracts the (additive) group from a ring.

Coercions are often inserted automatically by the system (hence the user doesn't type them) and typically not shown by Coq in its communication with the user. But the XML export of the Coq terms Mowgli deals with being at the "under the hood" level, they do show up in the terms Mowgli gets. Having no real significance for the "general mathematics" reader (as opposed to the "Coq user" reader) Mowgli aims at as an audience, these coercions are removed from the terms the proof-system-generic part of the system gets to see.

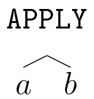
3.2.2 IDA-in-Helm enhancements

In the course of converting extracts from IDA, some rendering bugs were discovered and traced back to `d_c`, as well as the need to drop more coercions. Upon trying to understand what the existing code did to expand and correct it, the author realised that all the different bits of code treating different coercions seemed to actually aim at the same thing, except for simplifications the author of the code judged to be made possible by the type of the coercion, like e.g. the number of arguments it can be applied to. The latter assumption being proved to be sometimes incorrect (and the source of the observed rendering bugs), the treatment of all coercions has been factored into a generic template that always treats the "full generic case", without any simplifying assumptions.

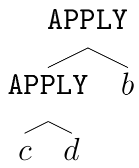
Additionally, coercions applied to functional values would sometimes give unreasonable results that would break expectations of code downhill in the processing chain. At the CICML level, function application is a tree whose root node is `APPLY`, the first child is the applied function and the subsequent children are the arguments to the function[1]. Here is for example the term corresponding to f applied to a and b (usually written as $(f\ a\ b)$ (or, more classically, $f(a, b)$):



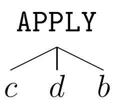
If f is a coercion, dropping that coercion means producing the term



Now, what if a is itself an `APPLY`? We then get in to this situation:



This is mathematically equivalent to



[1] The situation at the MathML Content level is very similar, with slightly different names being used)

This latter form is actually the one generally expected by rendering code, because it is the one created by Coq's CICML export. The `d_c` stylesheet has thus been adapted to produce the latter and not the former[2]. Alas, the algorithm to do that breaks a design choice of XSL, namely that a result cannot be re-used as input. As most current implementations provide an extension that "corrects" this design limitation, we chose to deviate somewhat from the standard and make use of this extension.

3.3 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 (in a teaching textbook)

> Many students at first believe that this and this is true, but be aware that it is not, as we will show through counter-example of vector spaces of infinite dimension, such as $\mathbb{R}[X]$, in section so and so on page so and so.

or (to take a real example from a real published book, namely [8])

> Moreover, the functors that are continuous are not difficult to describe.

or

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

The meaning of "many students at first believe", "not difficult to describe" or "in which we are interested" cannot be expressed in a theorem prover and escapes its checking framework, but they are still an important part of a mathematical document. And these bits of text speaks about and refers to formally defined mathematical notions ("continuous functors", "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.

This was previously not possible in the Helm system, because the only way a document could refer to a mathematical notion is by including its whole definition, as a separate paragraph, right there in the document. We have thus implemented the possibility to ask 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.

- Misses / contains some parts of the definition of the object that are included / not included by default, such as the name of the object, the body of the definition or the notation for the object.

These choices are all independent, although not all combinations make sense and will give a reasonable result. Syntactic shortcuts have been added for common combinations such as "only the notation for the object, in a way that integrates in a sentence", *i.e.* the notation for the object as a single word, without any paragraph break.

Let's note, however, that in the IDA-in-Helm effort the in-line rendering has been used also for statements that *do* have a precise mathematical meaning, but where some particular expression of it 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". Let's take a concrete example, the very first sentence of the chapter we treated:

[2]This in turn broke the expectations of some other rendering code, but this looked easier to fix than rewriting all the code to handle all different possible encodings of this construct.

> 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.

This says the same, mathematically, 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 to corresponding formal mathematical statement. Linguistically, however, several elements would have been lost there: "while" expresses an opposition that - while mathematically equivalent to a conjunction - is not equivalent as far as quality (e.g. ease of reading, clarity of point) of the document is concerned. A document making absolutely no use of that kind of subtleties of language would, indeed, be quite "dry" and hard to read for a human. Notice, too, 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 $\mathbb{Z}$".

One can theoretically imagine a Helm-like mathematical document production system having a sentence synthesis engine flexible enough that it can generate the (or a good part of the) full spectrum of sentences expressing the same mathematical statement without endangering safety of the generation[3], and thus removing the need for that kind of use of informal text about formal notions. First, that kind of perfect system seems to be a very long-term possibility, with no satisfying, usable implementation known yet. Second, it would still not remove the need for informal text about formal notions, and thus in-line rendering, like the first examples show.

3.4 Automatic generation of the notational stylesheets

3.4.1 Introduction

It is very advantageous to generate the notational stylesheets[4]. First of all it facilitates the user's work. Instead of writing the stylesheets in XSLT himself, and providing the system with all variations of templates that it needs to represent one single mathematical object in different contexts, he now has to write with each mathematical object only one description, concerning mostly *mathematical* properties of that object, in an easy XML language. From that single description all templates are generated automatically.

Secondly it makes it easier for the developer of the system too. Choices of style, for example "do not underline links in mathematical formulae", are made in the meta-stylesheets in a few places at most. So changes or improvements of these have to be done only there, and not in every single template of the stylesheets as it would have been the case without automatic generation.

3.4.2 First implementation for automatic generation

The first implementation for automatic generation of the notational stylesheets has been done by Pietro Di Lena and has clearly been recorded in [5] (in Italian). His work includes the definition of a simple XML language for the user, the definition of an intermediate XML language, used by the system, stylesheets from the user's input to the intermediate level, and meta-stylesheets from CIC-XML to MathML-Content, and from MathML-Content to both HTML and MathML-Presentation. He has also tested his implementation on a small sample of mathematical objects.

Example 3.1 *This is the code taken from the file* `set.xml` *written by the user, that defines the notation for intersection.*

```
<Symbols type="HTML">
  <symbol tag="intersect" priority="70" associativity="left">
    <case type="symbol" value="&#199;"/>
    <case type="unicode" value="&#8745;"/>
```

[3] that what the generated sentence says is indeed what the formal mathematical statement says

[4] The stylesheets responsible for generating the right notation for a specific notion, like e.g. $\mathbb{R}$ for the set of real numbers or $a + b$ for the addition of a and b

```
  </symbol>
</Symbols>

<Symbols type="MathMLPresentation">
  <symbol tag="intersect" value="&#x22C2;" priority="70" associativity="left"/>
</Symbols>

<Operator
   name   =  "INTERSECTION"
   uri    =  "cic:/Coq/Sets/Ensembles/Intersection.ind#element(/1/1)"
   cook   =  "true"
   arity  =  "2"
   p-tag  =  "intersect"
   h-tag  =  "intersect">
      <Notation type="MathMLContent">
        <m:apply helm:xref="\root-id">
          <i:op tag="intersect"/>
          <i:param id="1" mode="set"/>
          <i:param id="2" mode="set"/>
        </m:apply>
      </Notation>
</Operator>
```

The first two paragraphs have been written on the top of the file and define the appearance of the intersect-tag both for HTML and MathML-Presentation. The last paragraph is the description of the operator. The user has to give a name (`name`*), and the exact place where the operator has been defined (*`uri`*). The intersection has been defined inductively. That makes it necessary to add* `#element(/1/1)` *to the path. Would the user have wanted to define notation not for the intersection, but for its first constructor, then the last part of the uri should have been* `#element(/1/1/1)`*. Then the user says whether the operator has been defined inside a section (*`cook`*). That is the case here. The number of arguments is 2 (*`arity`*) and there are no implicit arguments. If there were, they could have been taken into account with the help of the hide-attribute. Then the system is told to use the newly defined intersect-tags when appropriate (*`p-tag` *and* `h-tag`*). Finally, the user describes what the MathML-Content should look like. Because there is no pre-defined tag for intersection, the user has to put some effort into this. Note that this description refers to the arguments of the operator.*

From this, stylesheets are made from CIC-XML to MathML-Content and from MathML-Content to HTML and MathML-Presentation, that handle the case that this operator is used inside the section of definition as well as the case that it is used outside.

This implementation has appeared very useful. It even covered some unforeseen uses (see 3.4.3). It has served as a core for further developments.

3.4.3 Changes

Although this implementation was well-considered, a big, thorough test was lacking. So when we tried to use it, we not only ran into some bugs, but we also encountered some situations of which the programmers had not thought. This forced us to carry some changes through. Besides that, we have made some changes and added some ideas purely for the convenience of the user. These adaptations affect the definition of the input XML language, the definition of the intermediate XML language, the meta-stylesheet from CIC-XML to MathMLContent, the meta-stylesheet from MathMLContent to HTML, and the arrangement of the input files.

We have not changed the meta-stylesheet from MathMLContent to MathMLPresentation. Most present-day browsers do not render MathMLPresentation in a satisfactory way. Moreover, the HELM system does not use the generated stylesheets from MathMLContent to MathMLPresentation. It uses some general hand-made stylesheets instead ([6]). So when time urged us to make some choices, we decided to ignore these meta-stylesheets.

Notation inside and outside sections As already shown in Example 3.1, the attribute `cook` is used to store information about whether the definition is made inside or outside a section. When a definition of, say, a mathematical operator is made inside a section and it is applied outside the section of definition, the CIC-XML of the applied operator looks different than when it would have looked if it were not defined inside a section. Because the section can contain variable declarations, the definition made within a section can rely on those variables, without explicitly mentioning them as arguments of the operator. Is this operator fully applied outside the section, then these variables get *instantiated*. The instantiation of variables is not seen as application.

Example 3.2
```
Section S1
    Variable A : CSetoid.
    Variable B : CSetoid.

    Definition product_inside :  CSetoid.
        <definition using A and B>
    Qed.
End S1.
Definition product_outside :  forall (A B:CSetoid), CSetoid.
    <definition using A and B>
Qed.

Section S2.
     Variable C : CSetoid.
     Variable D : CSetoid.

     Lemma iso :  (isomorphic (product_inside C D) (product_outside C D)).
End S2.
```
In the first section, `S1`*, two variables,* `A` *and* `B` *are declared. Then the definition of product between constructive setoids,* `product_inside`*, is given using these variables. Outside this section, another definition of product between constructive setoids is given (*`product_outside`*). The definitions are used similarly in* `Lemma iso` *in the section* `S2`*.*

The CIC-XML that codes the application of these operation in this lemma differs. The CIC-XML code that codes `(product_inside C D)` *is:*
```
<APPLY>
  <instantiate>
    <CONST uri="cic:/CoRN/algebra/CSetoids/product_inside.con"/>
    <arg relUri="S1/A.var">
      <VAR uri="cic:/CoRN/algebra/CSetoids/S2/C.var"/>
    </arg>
    <arg relUri="S1/B.var">
      <VAR uri="cic:/CoRN/algebra/CSetoids/S2/D.var"/>
    </arg>
```

```
  </instantiate>
</APPLY>
```

whereas the CIC-XML code that codes `(product_outside C D)` *is:*

```
<APPLY>
  <CONST uri="cic:/CoRN/algebra/CSetoids/product_outside.con"/>
  <VAR uri="cic:/CoRN/algebra/CSetoids/S2/C.var"/>
  <VAR uri="cic:/CoRN/algebra/CSetoids/S2/D.var"/>
</APPLY>
```

So this information is crucial for the correct generation of the stylesheets.

The user could say only *whether* a definition has been given inside a section. If this was the case there was no way to say on *how many* variables it relied. This means that it was not possible for the user to refer to these variables when he defined how this operator should be rendered. They were automatically rendered not unlike implicit arguments. This is a very practical solution, because the variables are never instantiated when one uses the definition in the same section it is defined in, and now the user could suffice with giving just one description of the rendering, that was used to generate the templates both for use inside the section and for use outside.

The result, however, was not very satisfactory. The choice to let a definition depend on variables seems to be influenced more by convenience and programming style, than by thoughts of the rendering. So we have changed the cook-attribute to have a numerical value. This also affected the meaning of the hide-attribute. At first the hide-attribute indicated the number of implicit arguments, now it indicates the number of implicit instantiations and arguments. (Implicit objects are always assumed to precede the explicit ones). Note that when the value of the cook-attribute is bigger than the value of the hide attribute, the user has to describe two different renderings: one for inside the section and one for outside, marked respectively by `cooked = "false"` and `cooked = "true"` (default).

Example 3.3 *The following describes a rendering for* `product_inside`*, assuming that* `\A`*,* `\B`*, and* `\times` *have already been defined. (MathMLPresentation has been omitted.):*

```
<Operator
   name   =  "PRODUCT"
   uri    =  "cic:/CoRN/algebra/CSetoids/product_inside.con"
   cook   =  "2"
   arity  =  "0" >
      <Notation type="MathMLContent" cooked="false">
        <m:apply helm:xref="\root-id">
          <i:op tag="times" definitionURL="\uri"/>
          <i:op tag="ci" definitionURL="\uri" bsymbol="A">A</i:op>
          <i:op tag="ci" definitionURL="\uri" bsymbol="B">B</i:op>
        </m:apply>
      </Notation>

      <Notation type="MathMLContent">
        <m:apply helm:xref="\root-id">
          <i:op tag="times" definitionURL="\uri"/>
          <i:param id="1"/>
          <i:param id="2"/>
        </m:apply>
      </Notation>

      <Notation type="HTML" cooked="false">
```

```
        \A \times \B
      </Notation>

      <Notation type="HTML">
        <o:param id="1"/>\times<o:param id="2"/>
      </Notation>
</Operator>
```

Now, `product_inside` *will be printed inside the section of definition as $A \times B$, and if the variables are instantiated outside the section, say by C and D, it is rendered as $C \times D$.*

Use of mathematical objects in various grammatical ways The examples we have seen so far are all quite complicated. Let us now have a look at the most basic example.

Example 3.4 *Assuming the plus-tag has already been defined for HTML and MathMLPresentation, the following defines the rendering for the addition operator:*

```
<Operator
   name   =  "PLUS"
   uri    =  "cic:/Coq/Init/Peano/plus.con
             | cic:/Coq/ZArith/BinInt/Zplus.con"
   arity  =  "2"
   c-tag  =  "plus"
   p-tag  =  "plus"
   h-tag  =  "plus"/>
```

From the information given in Example 3.4 all stylesheets are generated that take care of the expected rendering of the addition operator, provided that *it acts on two arguments*. So linguistically speaking, the stylesheets will only apply if the operator is the "subject" of the mathematical term. It will not apply, for example, to a term like `(commutative plus)`, because here `commutative` is the subject, and `plus` is acted upon. But even though `plus` is an "object" here, we would like to render this term as "+ is commutative". So we would like the stylesheets to apply.

To this end, we have extended the set of attributes of `Operator` with `object`, which has a boolean value. Default value is `object = "false"`, that will generate the same stylesheets as before; when `object = "true"` also stylesheets that apply in the object case will be generated. We have extended the set of attributes of `Notation` accordingly: to describe the notation in the object case, add `arity_0 = "true"`.

Observe that in the worst case, i.e. we want the stylesheets for the object case and our operator is defined within a section (see section 3.4.3), it can occur that the user has to give four descriptions for each type of notation (MathMLContent, MathMLPresentation, HTML).

Non-linear transitions Many times there is a 1-1 correspondence between the arguments in the CIC representation of a mathematical operator and the arguments in its final rendering. If this is the case, we call the transition *linear*. For instance, we would like to see "`(plus x y)`" as "$(x + y)$". In both representations the addition has two arguments and they are linked with each other in the obvious way.

It also quite often occurs that the final rendering contains less arguments than the CIC representation. This happens for example in the case of strict implicit arguments: one or more arguments are not printed, but they could be recovered from the arguments that are shown. For readability, we have chosen to also allow non-strict implicit arguments: we throw arguments away that can not be recovered from the remaining information. The function `cf_div`, for example, expects not only a field, a numerator and a denominator, but also a proof that the denominator is apart from zero. We do not need to record the field, because we can restore it from the types of the numerator and the denominator. So we take this as a strict implicit argument. But also the proof that the denominator is apart from zero we throw away, because this is not often seen as an argument in a standard mathematical text. This proof can not be restored from the remaining information. It is not sure that this is the right thing to do.

In our sample, it happens, though rarely, that we would like the final rendering to have more arguments than the CIC representation, i.e. we would like to see an argument more than once. For example the semi-group of constructive setoid functions, `FS_as_CSemiGroup`, takes only one element, that we like to render twice: "`(FS_as_CSemiGroup S)`" should be printed as "$S \to S$". And it was not possible to generate stylesheets to take care of this.

It was a technical problem that restrained us from generating these stylesheets. One of the generated stylesheets computes the length of the output string. In this stylesheet the length of each argument is stored in a variable. Such a variable took the name of the serial number in the CIC representation of the argument which length it holds. So, in our example, the variable that stores the length of `S`, would have been called `charcount_param_1`, because it refers to the first argument of `FS_as_CSemiGroup`. But because we want to see this argument twice, a variable with this name was also defined twice. It is not allowed in XSLT to define a variable in the scope of another variable with the same name.

We have solved this by giving the variable the name of the serial number in the HTML presentation of the argument which length it holds.

Notation for variables Although it was possible to generate stylesheets that took care of constants, inductively defined objects and even constructors, there was no way to do that for variables, too. But according to our experience, a nice rendering of variables increases the readability. Therefore we have adapted the meta-stylesheets to handle also variables.

Central definition of symbols As shown in example 3.1, the files from which the meta-styleheets take their data to generate the stylesheets, started with the definitions of the required tags: HTML tags for both symbol and unicode font and MathMLPresentation tags. These tags were used in the files that hold the stylesheets from MathMLContent to HTML and from MathMLContent to MathMLPresentation. So when we look in the file `html_set.xsl`, we see the following variables:

```
<xsl:variable name="set_intersect">
  <xsl:choose>
    <xsl:when test="$UNICODEvsSYMBOL = 'symbol'">&#xC7;</xsl:when>
    <xsl:when test="$UNICODEvsSYMBOL = 'unicode'">&#x2229;</xsl:when>
    <xsl:otherwise>???</xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="set_intersect_priority" select="70"/>
  <xsl:variable name="set_intersect">
    <xsl:choose>
      <xsl:when test="$UNICODEvsSYMBOL = 'symbol'">&#xC7;</xsl:when>
      <xsl:when test="$UNICODEvsSYMBOL = 'unicode'">&#x2229;</xsl:when>
      <xsl:otherwise>???</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
<xsl:variable name="set_intersect_priority" select="70"/>
```

(Note that the user has given the codes for the unicode and the symbol sign in decimal notation in this case, and that HELM has made hexadecimal notation out of it.) These variables are referred to only inside this file, as the names of the variables reflect.

The fact that the tags were only defined for the file they have been put in, was often a disadvantage.

Sometimes one needs the same symbol in two files and hence one had to define the same tags several times. To overcome this, we have put all tag definitions in one file, `HTML_SYMBOLS.xml`, and we have included this file – after a little transformation – in every other .xml-file.

So, an advantage of this approach is that the user needs only to define his symbols once. Or maybe even better: if we ourselves put the most common symbols in this file, say all LaTeX symbols with the names used in LaTeX, there would be hardly any need for the user to define any symbol at all.

There are also several disadvantages. Firstly, to include the symbols-file in every other .xml-file, we have used the XInclude package, which is an extension of XML that is not universally implemented. Secondly, at first an html_*.xsl-file contained only the tag definitions it really needed, and now an html_*.xsl-file contains *all* tag definitions. So the files have become bigger.

Both these disadvantages could probably be remedied by including the symbols in a much later state, in an XSLT file instead of in the XML files. To do this, another adaptation of the meta-stylesheets is required.

Natural language generation Although the generated stylesheets were only used to translate CIC-expressions to their usual mathematical symbolic rendering, for instance to replace "`plus`" by "+", the machinery seemed able to generate stylesheets to produce a bit of natural language too. So now the generated stylesheets 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`". To get the links right, we have made extensive use of the attribute "`bsymbol`". This is the XML description for the rendering of `commutes`:

```
<Operator
   name   =  "IS COMMUTATIVE"
   uri    =  "cic:/CoRN/algebra/CSetoids/commutes.con"
   cook   =  "1"
   hide   =  "0"
   arity  =  "1">
      <Notation type="MathMLContent">
        <m:apply>
          <i:op tag="csymbol" bsymbol="is,commutative,on">is_commutative</i:op>
          <i:param id="1"/>
          <i:param id="2"/>
        </m:apply>
      </Notation>

      <Notation type="HTML">
         <o:param id="2"/> \is  \commutative  \on
          <o:param id="1"/>
       </Notation>

</Operator>
```

where `\is`, `\commutative` and `\on` have been defined in the file `HTML_SYMBOLS.xml` as follows:

```
<symbol tag="commutative" value="commutative"/>
<symbol tag="is" value="is"/>
<symbol tag="on" value="on"/>
```

Besides that this is a rather onerous way to do something as simple as to print a part of a sentence, a drawback is that we have to use the non-breakable space ` `, since all breakable spaces are removed as part of the XSLT process. This severely limits the possibilities the user's browser has to break lines and in some cases leads to unbalanced line lengths or even a line being longer than the browser's window is wide.

At the moment, we have only generated natural language for mathematical operators. Logical operators are still presented symbolically. So we see "$\wedge$" and "$\rightarrow$", instead of "and" and "implies". Maybe these would have to change too. Then our presentation would be more uniform in style.

4 Discussion

4.1 Independent products

In mathematical vernacular, assuming an hypothesis and introducing an arbitrary element of a set is usually seen as two different concepts. But in type theory, the framework common to many theorem provers, they are folded into a unified concept, a Π-*product*. The difference between these two concepts is whether the product is *dependent* or *independent*. As the aim is that the output of helm be as close as possible to mathematical vernacular, these concepts have to be separated again at the rendering level, so that they can be rendered differently. This turns out to be extremely hard at this level in XSLT, and we thus rely on heuristics to approximate this decision.

4.2 Context dependent rendering

The rendering of mathematics in the document does not have a notion of before and after: mathematical objects are printed the same, independently of where they occur in the document. This does not seem to be always what one wants. In the theory of semi-groups, for instance, first the notion of being a unit is defined and 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 4.1 *$\forall S$:CSemiGroup.$\forall e$:S.$\forall f$:S. e is a unit of S$\wedge$f is a unit of S$\rightarrow$e=f*

(Note that this does not come close to the way it is formulated in the original IDA; there it simply reads "A semi-group has at most one unit".) But after we have proved the uniqueness, we would like to see "*the* unit" in all following uses.

What is more, web documents do not have to be linear like a book. It is well thinkable that there is more than one route through the document. This clouds the desired notion even more.

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.

4.3 Moving mathematical objects

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 is quite inconvenient. Because in this fashion the mathematics is completely separated from the description, there has to be another a way to connect the two. In HELM this link 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 of the way we work with C-CoRN. First, when we start to develop a theory, all objects are in our personal `devel` directory. And only later, when the development has been completed, everything is transferred to the fast core of C-CoRN. So, the HELM system imposes us to either write the XML description only after we have transferred everything, or to change all descriptions when we move the objects.

4.4 Using reflection

HELM tries to make readable proofs out of the lambda-terms of the proofs. In general it can be doubted whether this is a good decision and whether it would not have been a better idea to use the Coq-trees, which reflect the tactic script, instead. But in one case it is very clear that working from the lambda-term was not a good choice.

We have used in our formalisation some so-called *reflection* tactics. When one gives such a tactic a goal to prove, it will look at the *syntax* of that goal and tries to solve this goal by calculating whether it has a specific form. The lambda-term that such a tactic produces is in general unreadable and unlike anything that can be found in a mathematics book. We would also like to omit this part of the proof in the HELM rendering. It would not be missed, because mostly the "gaps" in the proof that thus arise do not exceed the steps that a human leaves out. However, this can not be done in HELM, because there is now knowledge about how parts of the lambda-term came to existence.

In the current HELM presentation we have tried to warn the reader by using angular brackets that some reflection is going on.

4.5 Use of a formal library: economy vs. coherence

This research has not only been a test case for HELM and the way it presents formal mathematics, but it has also been a test for the use of a formal library, especially C-CoRN, in a mathematical document. The choice to use the mathematics in the library has some huge advantages: you 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% from 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 any way, 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.

However, there are also disadvantages of using a library for this purpose. It seems to neglect the *coherence* of the document and it seems to destroy – in a weak sense – the existing coherence of the library.

A mathematical document usually answers 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 by the reader without any explanation, can be seen as a mistake. By using a library, we have paid no attention to this kind of logical coherence in the document. In the document we have put references to the library and it could well be that the order of the references does not meet the logical order of the corresponding items in the library. If we had not used a library, we could have taken the coherence into account. We could have started from scratch by making Coq files that follow the structure of the document.

Because Coq guards the logical coherence of the mathematics, the logical coherence in the document would be guaranteed too.

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.

Also C-CoRN has some kind of coherence: not only a formal coherence, but also an informal coherence, introduced by the documentation. Here I mean by "documentation" the text surrounding the formal objects; the part of a Coq file that is not checked. Most of the time this text explains some difficulties or particularities of the formalisation, but it also occurs, though not often, that the documentation and the division of the file reflect the original goal of the formalisation. In these cases it is very hard to add similar lemmas or even to use existing lemmas without changing the documentation. For example if one wants to add a lemma to a section and the documentation reads "The last lemma of this section is about ..." or one has used a lemma which is accompanied by a text saying that it is useful for a particular other lemma, one has to update the documentation. Although a good, informal explanation of the mathematical line of thought makes C-CoRN nice to read, there seems to be some tension between having a documentation geared to the mathematical content, and the reusability of the library.

5 Conclusions

5.1 Analysis

The process of converting IDA to a Helm document consisted essentially in three activities:

- Formalising the mathematics of IDA in Coq.
- "Porting" IDA to the Helm framework.
- Enriching the Helm framework so that it could accommodate IDA.

While the first is essential to any approach based on formal mathematics, it is interesting to analyse how much in the Helm enrichment was specific to IDA or how much was a generically useful improvement. Also, it is interesting to see how much of the IDA "porting" was reasonably to be expected (let's say similar to importing a document from one word processor to another word processor) and how much would have been a stretch of mind even for a document written from scratch for the Helm framework.

Most of the changes to Helm were actually quite generic. The only somewhat ad-hoc thing is the "breadcrumbtrail" feature: a way to import IDA's frame-based navigational system into Helm, which has its own frame-based navigational system. As frame-based navigational systems are rather widespread among web documents, even this feature is rather generic.

The Javascript code in IDA needed some very minor adaptations[5]: it is generating links dynamically (among others for images that change depending on the answer given in multiple-choice exercises) and these had to be changed to make use of the `xml:base` tag the getter put in the document. The pages of IDA themselves were old-style HTML, but Helm requires XHTML; this conversion was done automatically by [7]. Again, these changes, while not all exactly expected, are not unreasonable.

5.2 Future Outlooks and remaining problems

During the work on IDA-in-Helm, there has been a continuous cooperation between the Nijmegen and Bologna team to solve small bugs in Helm and make alterations. This has improved the system a lot.

[5] There are exactly three lines out of about 380 that were changed

On a higher level, it is clear that we don't have the ideal tool for writing integrated documents yet. For example, the document processing pipe line gives a feedback that is too long: only at the very end the author can judge whether his alterations take the desired effect. Furthermore, the www/html/xml platform is not yet mature enough for dealing with interactive mathematical documents, e.g. because MathML is poorly supported, so a really nice rendering of mathematical formulas is not possible. It works quite well for distributed libraries, but not so well for documents that should basically be read by a student on his own computer.

5.3 Conclusion

Let's now come back to the points that have been raised in the Introduction Section 1. We have achieved the goals that we set ourselves: the statements and proofs of IDA are rendered by the Mowgli tools out of the formal C-CoRN content and the hyperlinks in IDA are also generated out of the formal semantical links in C-CoRN. So in that extent, the Mowgli tools are certainly successful: this prototype translation of C-CoRN through Helm in IDA shows that these things are possible.

With respect to the quality of the rendered statements and proofs: The proofs are fairly good. They contain quite a lot of detail (and therefore we have added some additional features for hiding information, like leaving assumptions implicit), but the structure is clear and good and the folding/unfolding of subproofs (to inspect a proof in finer detail) works very well. Proofs that are automatically generated (using a decision tactic of Coq) become very ugly, but that's unavoidable. (These proofs should be rendered in a special way, disabling the possibility to inspect the proof.) The lemmas are also good, but the definitions are sometimes a bit too Coq specific, containing λ's and words like "Record" and "Case". But this is also partly a matter of choice: either one wants to cater just for students (and then one could introduce a rendering that suppresses the word "Record"), or one also wants to show, e.g. to Coq users, what *exactly* the definition is, and then we want to see "Record" with the names of the labels that are used. The "Record" case is an interesting example in this respect, because, e.g. in the Record type that describes semi-groups, there is the property of "being a right unit", which has a name (a record label) "runit", that we certainly don't want to suppress, because it is referred to later in proofs. So the case of what to suppress and what not is not so clear and can't be made generically.

With respect to the user-friendliness, the project has generated useful meta-stylesheets that can be instantiated to generate stylesheets that do the transformation. These meta-stylesheets can be used very well by persons that know HTML, MathMLContent and MathMLPresentation, which may seem like a heavy requirement, but it's unavoidable. (If you want to write nice LaTeX symbols, you also have to know the LaTeX codes.) For an author who is Coq-knowledgeable, it would be more interesting to let the MathML be extracted from the Coq file. This is also possible, if one uses coqdoc to add documentation information to Coq files.

References

[1] `http://corn.cs.kun.nl/`

[2] A. Cohen, H. Cuypers, H. Sterk, *Algebra Interactive!: learning algebra in an exciting way*, Springer 1999.

[3] L. Cruz-Filipe, *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*, PhD thesis, Nijmegen 2004.

[4] L. Cruz-Filipe, H. Geuvers and F. Wiedijk, C-CoRN, the Constructive Coq Repository at Nijmegen. In: Andrea Asperti, Grzegorz Bancerek, Andrzej Trybulec (eds.), *Mathematical Knowledge Management, Proceedings of MKM 2004*, Springer LNCS 3119, 88-103, 2004.

[5] P. Di Lena, *Generazione automatica di styleheet per notazione matematica*, Master's thesis, University of Bologna, 2002.

[6] I. Loeb, *Presentational Stylesheets*, MoWGLI deliverable, 2003.

[7] HTML tidy by Dave Raggett, ©1998-2000 World Wide Web Consortium, `http://tidy.sf.net/`

[8] M. Niqui, *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*, PhD Thesis, Radboud Universiteit Nijmegen, September 2004, section 4.2, page 98.