INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Prototype n. D4.c
Prototype functionalities for assisted annotation

Main Authors:
Philippe Audebaud, Luca Padovani

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 Overview

This document describes the architecture of the tool for assisted annotation and some of the issues related to it. The tool allows the user to provide alternative presentations for fragments of COQ proofs via a graphical interface based on the TeXmacs [vdH] structure editor.

2 Introduction

The MoWGLI transformation mechanism aims at providing a readable presentation for documents of formalized mathematics exported from the COQ library. During the transformation, the CIC terms contained in the mathematical objects to be presented are structurally rearranged and finally mapped into a suitable presentation markup (either plain HTML or HTML with embedded MathML [ABD$^+$01, ABD$^+$02] or pure MathML).

Although the objects of the COQ library are presented using "natural language", they are still far from the corresponding presentation that could be found in a textbook. This is no surprise because:

- the complete formalization introduces a large number of details which the human usually disregards;
- the structure of the proof may be innatural because of the way it was created.

The mechanism for progressively unfolding sub-parts of the proof can be useful for resolving parts of the first issue. However the same mechanism is limited by the second issue: if the proof is not well-structured, or it is structured in a way that differs considerably from the "intuitive" presentation of the same proof, human intervention is required.

In this document we present an annotation mechanism which allows the human to modify and rearrange to a certain extent the presentation of the proof. The idea is that we can exploit the structured nature of the documents to easily identify parts that we want to annotate, that is parts for which we want to provide an alternative presentation when the one that is automatically generated is not satisfactory. The annotation is a piece of markup that *replaces* a fragment of the document in its presentational form. Annotations may have *holes*, that is placeholders for subformulas and/or subproofs occurring inside the annotation that we do not want to change.

3 The tool for annotations

The annotation of a document involves the use of an interactive tool with the following characteristics:

- the tool must allow some form of *structured editing*, where the structure of the edited document is used for constraining the kind of modifications the user is allowed to do;
- the tool must be able to display and edit mathematical proofs and formulas.

We found that TeXmacs [vdH] satisfies both requirements quite naturally, hence the prototype has been developed around it. The development went through the following major steps:

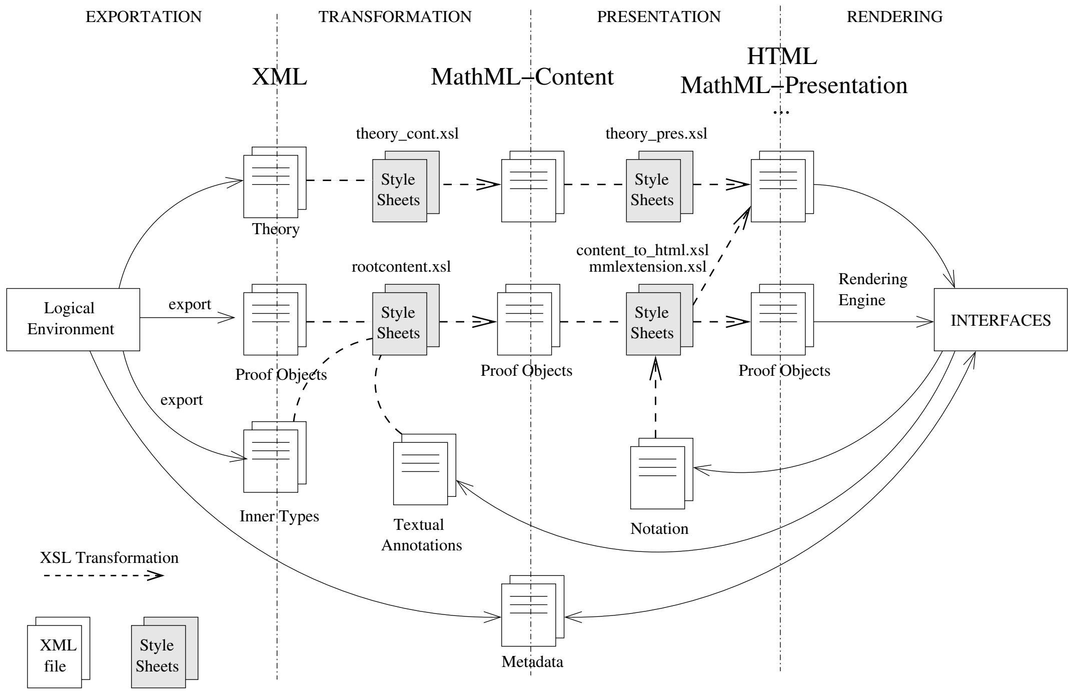


Figure 1: Transformation Phases and main stylesheets.

- implementing an XML import/export mechanism for TeXmacs (until very recently TeXmacs was able to save documents either in some purely presentational form like LaTeX or non-XML, dedicated markup);
- forking the transformation pipeline of Figure 1 so that the TeXmacs XML format is supported along with HTML and MathML formats;
- recognizing fragments of the edited document corresponding to annotations;
- updating the stylesheet pipeline to take annotations into account.

These phases will be detailed in the next subsections.

3.1 XML format for TeXmacs documents

Since the original TeXmacs distribution did not provide support for importing and exporting TeXmacs documents in XML format, and since our project is heaviliy based on XML, the first step was to make TeXmacs XML-aware. We did so by creating a TeXmacs plugin consisting of:

1. a binding to a binary library (MIXP [Bez]) for parsing XML documents;
2. a set of Scheme functions[1] for creating an internal TeXmacs document from its XML form;

[1]TeXmacs uses Scheme as language for implementing extensions to the editor.

3. a set of Scheme functions for pretty-printing a TeXmacs document into XML form;
4. a set of menu items of importing/exporting TeXmacs documents from/to XML form.

From now on we will use the name TMXML to denote the XML format of TeXmacs documents. An example of TMXML markup will be given in Section 5.1.

3.2 TeXmacs presentation of COQ proofs

This phase involved the implementation of a new transformation pipeline from the intermediate representation of proofs to the final presentation.

As it is detailed in deliverables D2.c, D2.d, D2.e, D2.f, we can distinguish two main phases in the transformation process of a document: the first transformation converts the low-level, formal information into content-oriented information which is not tied to CIC; the second transformation converts the intermediate representation into the final presentation (see Figure 1). As already pointed out, the intermediate representation is useful for modularization purposes because it reduces the number of stylesheets to be produced: the stylesheet for obtaining the intermediate representation can be written once and for all, no matter what the target markup language will be. The advantage clearly increases as soon as we use the same intermediate representation for mapping several logical dialects. Apart from the purposes specific to the implementation of the prototype it is interesting to notice that there is a subphase of the transformation from the intermediate representation to the final presentation that is actually common no matter what the final presentation markup is. This phase has to do mainly with the decision on where to put line-breaks in the proof in natural language. Then, depending on whether we are aiming at HTML or MathML, a suitable markup that achieves the wanted line-break is generated. In the case of HTML we will normally use `<br/>` elements to break lines and then a sequence of non-breakable spaces for indentation. With MathML it is more natural (and wiser) to use tables. TMXML is somehow in between as it combines the linearity of HTML with the structure of MathML.

Thus we have introduced another level in the transformation pipeline which is used to represent the abstract presentation of a proof in terms of *boxes* for horizontal material (that is material that is to be presented on a single horizontal line), vertical material (that is material that has to be presented vertically), and indentation. The “material” consists of either other boxes or leaf nodes, which in our case can be made of plain text or mathematical formulas. The language is called BoxML and takes its name from a similar language [dJ00, vdBV94] we took inspiration from. In Appendix A we give the complete DTD of the language of boxes that we are currently using, along with a detailed semantics of its tags.

The advantage of having a further level is that the transformation from the intermediate representation to the level of boxes, which is relatively complex since it is responsible for passing from the content-oriented document to its presentation oriented form, is done once and for all. Then, starting from the boxed document, the trasformations towards HTML, MathML, and TMXML are straightforward.

But the language of boxes plays also another role which is specific to the annotation mechanism. In fact, we had to decide what markup language to use for representing annotations which are presentation-oriented by definition. We could use either one of HTML, MathML, or TMXML, but doing so would likely make the annotation useless in case the user is interested to view the document in a presentation format which does not match the format in which the annotation is saved. By imposing a few restriction on what an annotation can be, for example by constraining the user to only reuse formulas or sub-formulas in the original document, we were able to use BoxML as the format for annotations.

3.3 Finding annotations

Annotating a proof implies modifying the document seen by the user. Once the annotations are in place, they have to be extracted from the document and stored in a suitable resource. There are two possible ways for finding annotations in the modified document:

- while annotating, the user is also required to mark the annotations with a flag that will be used in the annotation extraction phase for detecting annotations, or
- a mechanism is provided which understands automatically where the annotations are, by looking at the differences between the source document (the document imported initially into TeXmacs) and the final document (the document exported from TeXmacs).

We have experimented with the second possibility since it puts less burden on the user and also it fits better in our architecture since not every fragment can be annotated and the user may not be aware of what can or cannot be annotated.

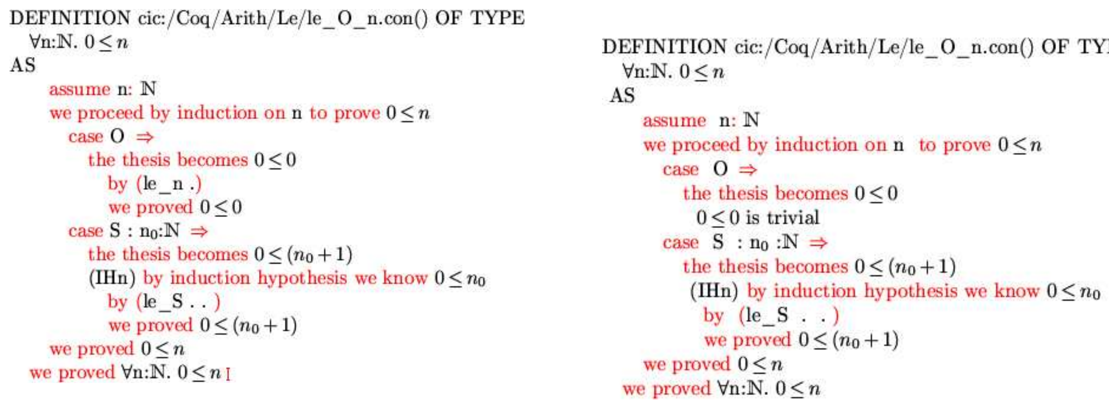
The extraction of annotations is thus reduced to finding parts of the documents that have changed with respect to the original document. This task can be easiliy achieved by means of an XSLT stylesheet whose input are the two documents (the original one and the modified one), and whose output is an XML document listing the sequence of annotations found.

It is essential in order for the diffing algorithm to work properly that document fragments that can be annotated are marked somehow. To this aim we have provided a special macro `box:g` which accepts two arguments: the first argument is the identifier of the fragment, the second argument is the fragment itself. Only the second argument is actually displayed, whereas the first argument remains hidden and is only used by the diffing algorithm.

The following fragment of TMXML markup shows that the fragment with identifier `i88` can be annotated:

```
 <macro name="box:g">
  <paragraph> i88 </paragraph>
  <paragraph>
[...]
  </paragraph>
 </macro>
```

The stylesheet computing the difference between the original and the modified documents recovers any fragment that is marked as "annotatable" in the original document, it finds the corresponding fragment (the fragment with the same identifier) in the modified document and compares the two fragments node by node. The comparison is not properly recursive in the sense that whenever an annotatable sub-fragment is found inside another annotatable fragment, the sub-fragment is not compared against the corresponding sub-fragment in the modified document. Instead, a reference to the sub-fragment is output thus leaving a *hole* in the annotation which will be filled in during the next presentation of the document.



```
DEFINITION cic:/Coq/Arith/Le/le_O_n.con() OF TYPE
  ∀n:ℕ. 0 ≤ n
AS
    assume n: ℕ
    we proceed by induction on n to prove 0 ≤ n
      case O ⇒
        the thesis becomes 0 ≤ 0
          by (le_n .)
          we proved 0 ≤ 0
      case S : n0:ℕ ⇒
        the thesis becomes 0 ≤ (n0 + 1)
        (IHn) by induction hypothesis we know 0 ≤ n0
          by (le_S . . )
          we proved 0 ≤ (n0 + 1)
    we proved 0 ≤ n
  we proved ∀n:ℕ. 0 ≤ n
```

(a)

```
DEFINITION cic:/Coq/Arith/Le/le_O_n.con() OF TYPE
  ∀n:ℕ. 0 ≤ n
AS
    assume n: ℕ
    we proceed by induction on n to prove 0 ≤ n
      case O ⇒
        the thesis becomes 0 ≤ 0
          0 ≤ 0 is trivial
      case S : n0 :ℕ ⇒
        the thesis becomes 0 ≤ (n0 + 1)
         (IHn) by induction hypothesis we know 0 ≤ n0
          by (le_S . . )
          we proved 0 ≤ (n0 + 1)
    we proved 0 ≤ n
  we proved ∀n:ℕ. 0 ≤ n
```

(b)

Figure 2: The theorem $\forall n \in \mathbb{N}.0 \leq n$ before and after being annotated.

3.4 Viewing annotated documents

The transformation pipeline had to be changed in order to support annotations. In particular, for each document fragment processed a check is made first to verify whether there exists an annotation whose identifier matches the identifier of the fragment being processed. In case there is no annotation the processing resumes as in the original transformation pipeline. If there is an annotation, the markup present in the annotation is copied in the result document, and the holes replaced with the result of processing the referenced sub-fragments.

In Section 5 we discuss some of the issues in this phase that still need further investigation.

4 An example

In this section we show the annotation mechanism at work on a necessarily simple proof. We consider the proof `cic:/Coq/Arith/Le/le_O_n` stating that any natural number is greater than or equal to zero. Figure 2(a) shows the proof as it is generated without annotations (the screenshot has been taken from the TeXmacs window). Then we have slightly changed the base step of the inductive proof by removing the justification and stating that the base case is trivial.

The result, once exported from TeXmacs, is compared with the original document and the following annotation is produced (only the relevant fragments are shown):

```
<?xml version="1.0"?>
<Annotations xmlns:box="http:/helm.cs.unibo.it/2003/boxml"
             of="cic:/Coq/Arith/Le/le_O_n.body">
  <Annotation of="i88">
[...]
    <box:text>&#xA0;</box:text>
[...]
    <box:text>&#xA0;</box:text>
    <recurse-pure id="i89"/>
    <box:text>&#xA0;</box:text>
    <box:text>is trivial</box:text>
    <box:text>&#xA0;</box:text>
    <box:text/>
  </Annotation>
</Annotations>
```

There is one annotation only for the document fragment with identifier `i88`. The annotation consists of some text, manily for indentation, and a sub-formula $0 \leq 0$ occurring at some point. Note that the subformula is not embedded directly in the annotation. Rather, a placeholder is used instead, pointing at the document fragment with identifier `i89`. The name `recurse-pure` indicates that the placeholders stands for a formula, while the name `recurse-proof` indicates that it stands for a sub-proof.

The same proof, re-processed with the annotation, is shown in Figure 2(b).

5 Issues

There are some important issues that remain to be investigated and eventually fixed. We discuss them in the next two sections.

5.1 Notation for TeXmacs

The transformation that generates TMXML markup differs from the ones generating MathML or HTML in that TMXML markup tends to be more "content-oriented". By this we mean that the structure of the document imported in TeXmacs has emphasis over the content rather than the presentation of the information. Then, it is TeXmacs' duty to provide the appropriate notation by means of *styles* and possibly some Scheme code associated to them. In order to clarify this point we can have a look at the following fragment of TMXML document which encodes the formula $\forall n : \mathbb{N}.0 \leq n$:

```
<macro name="prodc-1">
 <paragraph>
  <macro name="param">
   <paragraph> n </paragraph>
   <paragraph>
    <macro name="apply">
     <paragraph> refc </paragraph>
     <paragraph> nat </paragraph>
     <paragraph/>
     <paragraph> cic:/Coq/Init/Datatypes/nat.ind </paragraph>
    </macro>
   </paragraph>
  </macro>
 </paragraph>
 <paragraph>
  <macro name="apply">
   <paragraph> appr </paragraph>
   <paragraph>
    <macro name="apply">
```

```
     <paragraph> refc </paragraph>
     <paragraph> le </paragraph>
     <paragraph/>
     <paragraph> cic:/Coq/Init/Peano/le.ind </paragraph>
    </macro>
   </paragraph>
   <paragraph/>
   <paragraph/>
   <paragraph>
    <macro name="apply">
     <paragraph> refc </paragraph>
     <paragraph> O </paragraph>
     <paragraph/>
     <paragraph> cic:/Coq/Init/Datatypes/nat.ind </paragraph>
    </macro>
   </paragraph>
   <paragraph> n </paragraph>
  </macro>
 </paragraph>
</macro>
```

The constructs encoded in the markup do not explicitly refer to a particular presentation. For instance, the "forall" construct is a macro with name `prodc-1` taking two arguments, the first being the bound variable, the second being the body of the construct in which the bound variable occurs. The TeXmacs style that deals with the universal quantifier, which instruct TeXmacs to render the construct as $\forall\langle binding\rangle.\langle body\rangle$, is the following:

```
<assign|prodc-1|<macro|a|b|\
    <apply|forall><arg|a>.<format|line break> <arg|b>>>
```

In turn, the binding is made of a variable along with its type. In TMXML this is encoded using the `param` macro which has the following style associated:

```
  <assign|param|<macro|x|y|<arg|x>:<arg|y>>>
```

This meachanism for specifying notation is extremely flexible but it is also completely unrelated to the mechanism for notation that we are using in the transformation pipeline. In practice, when we transform a document where the output format is TMXML we are ignoring the notation provided by our stylesheets because in fact we rely on the notation machanism of TeXmacs. This poses a problem since in order to consistent notations between the different output formats we need to keep the XSLT stylesheets and TeXmacs styles in sync. Since we are already using a technique for generating automatically XSLT stylesheets for mathematical notation starting from a higher level description of the notation itself, it might be the case that a simple extension to this mechanism will allow us to also generate automatically the corresponding TeXmacs style (that is, the style that will give exactly the same output).

5.2 Smooth integration of annotations in the pipeline

The idea of using document fragment identifiers for associating annotations is clean in principle, but it poses a number of problem in practice.

First of all, identifiers should be unique: this requirement is essential when annotations are combined with the document during the processing. However, as Figure 1 shows, the final document results from the combination of two parts, the exported CIC terms and the inner types, and it may be the case that the same identifier is used in both parts, even though it obviously refers to two unrelated document sub-trees. Hence there is the possibility that the identifier used in the hole of an annotation is ambiguous. One way to fix this problem would be to use different schemes of identifiers, depending on what kind of document the identifiers should point to. Alternatively, the holes in the annotations must be enriched with some piece of information that disambiguates the identifier.

The second problem has also to do with processing of holes inside annotations. In fact, the presence of a hole in the annotation should re-activate the normal transformation on the referred subtree. For formulas this is not a problem, because the transformation of formulas is context-free, that is there is no contextual parameter or mode that affects the way formulas are rendered. However, holes for sub-proofs are much more delicate: the XSLT templates processing them have a number of different modalities and some of them accept attributes affecting the way they are transformed. If no annotations are present, these parameters are naturally computed by the stylesheet during the traversal of the document. When annotations are present, however, there is a gap between the point where the annotation was inserted and the location of the hole. In this case it may be not so simple to compute the updated set of parameters to be passed when recurring at the location of the hole, nor it is clear how to recover this information without constraining some more the way annotations are used.

A The language of boxes

What follows is the DTD of the language of boxes that we are currently using. Table 1 gives an intuitive semantics for the elements of the language.

```
<!ENTITY % box "h|v|i|text|obj|action">

<!ELEMENT box (%box; | g)>

<!ELEMENT h (%box; | g)*>

<!ELEMENT v (h | i)*>

<!ELEMENT i (h)>
<!ATTLIST i by CDATA #REQUIRED>

<!ELEMENT text (#PCDATA)>
<!ATTLIST text color NMTOKEN #IMPLIED>

<!ELEMENT obj ANY>
<!ATTLIST obj obj_indent NMTOKEN #IMPLIED>

<!ELEMENT g (h | i)*>
<!ATTLIST g helm:xref NMTOKEN #IMPLIED>

<!ELEMENT action (%box;)*>
<!ATTLIST action type NMTOKEN #REQUIRED>
```

References

[ABD$^+$01] Ron Ausbrooks, Stephen Buswell, Stéphane Dalmas, Stan Devitt, and Angel Diaz et al. Mathematical Markup Language (MathML) Version 2.0 Specification. W3C Recommendation. `http://www.w3.org/TR/MathML2`, February 2001.

[ABD$^+$02] Ron Ausbrooks, Stephen Buswell, Stéphane Dalmas, Stan Devitt, and Angel Diaz et al. Mathematical Markup Language (MathML) Version 2.0 (2nd Edition) Working Draft. `http://www.w3.org/TR/2002/WD-MathML2-20021219/`, December 2002.

[Bez] Thierry Bezecourt. Mixp - A Guile interface to expat. `http://www.thbz.org/mixp/`.

[dJ00] M. de Jonge. A pretty-printer for every occasion, 2000.

[vdBV94] M. van den Brand and E. Visser. From box to tex: An algebraic approach to the construction of documentation tools, 1994.

[vdH] Joris van der Hoeven. GNU TeXmacs. `http://www.texmacs.org/`.

Element	**Attributes**	**Description**
`box`		Root element for a boxed document.
`h`		Horizontal box. Its children are displayed on the same line.
`v`		Vertical box. A new line is begun after every child.
`i`	`by` is the amount of indentation from the left margin	Indent the content of the box by the specified amount.
`text`	`color` is the optional color for the text to be displayed	Display a string of plain text.
`obj`	`obj_indent` is the amount of indentation from the left margin to be passed when processing the embedded object	Embed an object in the boxed document. The object is usually encoded in a markup different from BoxML. Typically it is either MathML or TMXML.
`g`	`helm:xref` is a pointer to the corresponding subtree at the content level	This element has no rendering semantics (it is "transparent"). Its only purpose is to preserve at the level of boxes the relevant structure so that it is possible to implement forms of semantic selection when the target markup allows it.
`action`	`type` is the type of action. Currently only `toggle` is supported	Represents a number of alternative renderings. This is used for implementing the folding/unfolding mechanism of subproofs. Depending on the target markup language (MathML, HTML, or TMXML) this element will be mapped to the appropriate markup: for MathML and TMXML there are specific elements to enable toggling. For HTML we generate JavaScript code that modifies the document DOM tree.

Table 1: Informal semantics of the language of boxes.

—