INFORMATION SOCIETY TECHNOLOGIES
(IST)
PROGRAMME

Project IST-2001-33562 MoWGLI

Prototype n. D2.c - D2.d
Stylesheets to intermediate representation

Main Authors:
Andrea Asperti, 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 Overview

This document provides a quick reference to the transformation stylesheets which have been developed for passing from the low-level XML description of the library of the COQ Proof Assistant to a suitable intermediate representation, discussed in the companion document D2b. Comprehensively, for this phase of the transformation process, we developed more than 5000 lines of XSLT, organized in about 20 stylesheets.

2 Introduction

In the Preliminary Report on Application Scenarios and Requirement Analysis, we already pointed out the use of XSLT stylesheet as the most natural approach for the re-mathematization of the logical, symbolic content of the information. The broad goal was that of developing coherent and well maintained libraries of notational stylesheets, publicly available on the web and freely reusable by any interested party.

Form the architectural point of view, we identified four different transformation phases of a document from the internal representation in some Proof Assistant Application to its final rendering: exportation, transformation, presentation, and rendering. Fig. 1, taken form Report D1a, gives a sketch of the overall architecture of these phases.

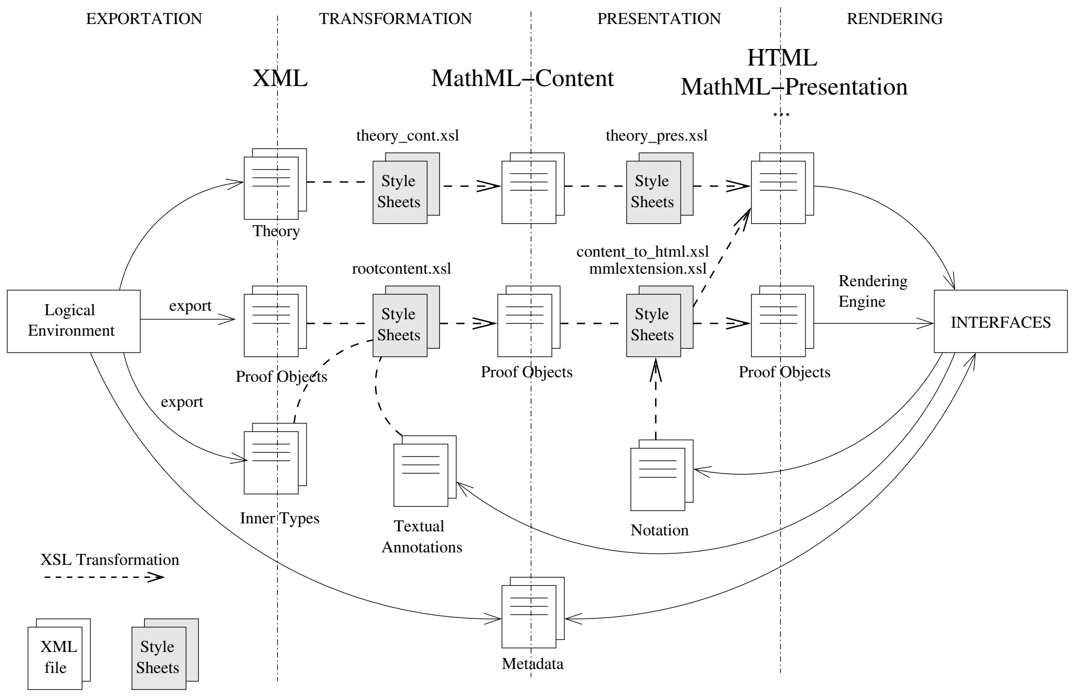

Figure 1: Transformation Phases and main stylesheets

The first phase (exportation) has been already and successfully completed (see deliverable D2a). This report is concerned with the second phase, and in particular with the transformation of mathematical items and its sublevels: namely formulas, terms and proofs. This phase is the core of the transformation process, and required a deep investigation of the intended intermediate representation, discussed in the companion document D2b. Its main purpose is that of improving the modularity of the whole architecture, decoupling the purely presentational and notational aspects.

As described in Figure 1, the stylesheets are organized in two main pipelines, one starting from documents describing the macrostructure (views, or theories), and one starting from documents describing single mathematical items.

The pipeline for the macrostructure is not particularly complex: our documents are essentially XHTML files "wrapping", via a suitable markup, the logical items. The transformation process is merely in charge of recursively applying the transformations to each individual item. For this reason, we shall merely concentrate on the stylesheets in charge of the latter transformations, in the following.

3 A transformation example

The files of our library have been automatically extracted from the Coq proof-assistant library. Thus they are XML files valid against the DTD we proposed for the Calculus of (Co)Inductive Constructions (CIC), which is the logical framework Coq is based on. According to the Curry-Howard isomorphism, CIC can be seen as both a lambda-calculus and a logic, where there is no syntactical distinction between terms and types. As terms of a lambda-calculus, the terms of CIC can represent either mathematical formulae (if they are used as types) or programs (when they are really used as terms). As terms of a logic, the terms of CIC can represent either statements (if they are used as types) or proofs (when they are really used as terms). Thus CIC can be used as an unified framework to reason about mathematical facts and their proofs and program specifications and their implementations.

Even if there is no syntactic distinction between terms and types and no syntactic clue about the intended usage of a term, the typing rules of CIC can be used to determine the exact status of a term:

- A term of type the sort `Prop` is a proposition (a mathematical statement).
- A term of type the sort `Set` is a datatype. E.g: integer numbers, lists, sets, morphisms, etc.
- A term whose type is of type `Prop` is a proof, i.e. a witness of a proposition (which is its type).
- A term whose type is of type `Set` is an element of a datatype. E.g.: 0, the empty list, the identity function, etc.

In what follows we will say that a term t is of sort S when the type of t has type S.

Among other things, we are interested in explicitly recover the distinction between the four preceding classes at the content level. In particular, we can use MathML Content to encode both datatypes, their elements and mathematical statements. Since MathML Content provides no markup to encode proofs and proof steps, we have developed (as the deliverable D2b) a new OMDoc module supporting the specification of proofs.

In Fig. 2 we present a very small proof in CIC of the fact that for all natural numbers n holds $0 \leq n$ where $\leq$ is axiomatically defined by the two following rules:

$$\begin{array}{ll} le\_n: & \forall n. n \leq n \\ le\_S: & \forall n, m. n \leq m \Rightarrow n \leq m+1 \end{array}$$

1. $\lambda n : nat.$
2. $(nat\_ind$
3. $\lambda m : nat.(le\ 0\ m)$
4. $(le\ n\ 0)$
5. $\lambda m : nat;\ H : (le\ 0\ m).(le\_S\ 0\ m\ H)$
6. $n)$

Figure 2: A small proof in CIC.

The proof starts assuming a generic but fixed natural number n (line 1) and then proceed by induction over n (lines 2-6). Proceeding by induction means applying the induction principle over natural numbers (nat_ind, line 2) to the proposition we want to prove ($\lambda n.0 \leq n$, line 3), the recursive argument (n, line 6), and the two proofs of the base case and the inductive step (lines 4 and 5): indeed $(le\_n\ n\ 0)$ has type $0 \leq n$ (i.e. is a proof of the base case $0 \leq n$) and $\lambda m : nat;\ H : (le\ 0\ m).(le\_S\ 0\ m\ H)$ has type $\forall m : nat.0 \leq m \Rightarrow 0 \leq m + 1$ i.e. it proves the inductive step.

Fig. 3 shows the same proof in XML. We can briefly remark that during the exportation phase we made explicit part of the information that was implicit in Fig. 2. In particular:

1. Each sub-expression is marked with its sort, i.e. the type of its type (XML attributes `sort`). The sort of each type was computed during the exportation phase together with its type. It is this sort information that allows us to discriminate expressions from proofs during the transformation to MathML Content.

2. Each variable occurrence (XML elements `REL`) now refers explicitly to its binder (XML elements `idref` and `value`, which is the DeBrujin index of the variable).

3. Each definition or lemma occurrence (XML elements `CONST`, `MUTIND` and `MUTCONSTRUCT`) carries the URI of its definition or proof.

4. Each sub-expression is given an unique identifier (XML attribute `id`) that is used to attach information to it in an external file. In particular, in a separate file we store the logically redundant type of each expression of sort `Prop` (i.e. the thesis $0 \leq n$ of the base case and the thesis $\forall m : nat.0 \leq m \Rightarrow 0 \leq m + 1$ of the inductive step). The two files (the proof and the types) will be merged during the transformation to the content level.

Finally, Fig. 4 shows the OMDoc file we obtain applying to the previous XML file our transformation. To make it more understandable, all the MathML Content sub-trees have been replaced with their LaTeX rendering. All the elements left belong to the OMDoc markup.

The root element is an `omdoc:proof` (lines 2-53). The proof is made of just one big `omdoc:derive` step (lines 2-52) that concludes that $\forall n : nat.0 \leq n$ (lines 6-7) by discharging (line 8) the declaration of the arbitrary but fixed n of type nat (lines 10-14). More precisely, the `omdoc:method` `Intros+LetTac` is used to generalize the thesis of its only sub-proof (lines 9-50) by discharging all the variables and hypotheses in its context (i.e. all its `omdoc:declaration` and `omdoc:hypothesis` children). In our example, the sub-proof starts with the declaration (lines 10-14) and is concluded by a final `omdoc:derive` step (lines 15-49) that proves that $0 \leq n$ (lines 16-17) by induction over n (line 18). Unsurprisingly, the proof by induction has two subproofs (lines 19-27 and 28-47) which correspond to the base case and inductive step.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE ConstantBody SYSTEM "http://mowgli.cs.unibo.it/dtd/cic.dtd">

<ConstantBody for="cic:/Coq/Arith/Le/le_O_n.con" params="" id="i162">
  <LAMBDA sort="Prop">
    <decl id="i0" type="Set" binder="n">
      <MUTIND uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0" id="i7"/>
    </decl>
    <target>
      <APPLY id="i9" sort="Prop">
        <CONST uri="cic:/Coq/Init/Datatypes/nat_ind.con" id="i120" sort="Prop"/>
        <LAMBDA sort="Type">
          <decl id="i102" type="Set" binder="m">
            <MUTIND uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0" id="i106"/>
          </decl>
          <target>
            <APPLY id="i108" sort="Type">
              <MUTIND uri="cic:/Coq/Init/Peano/le.ind" noType="0" id="i114"/>
              <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"
                noConstr="1" id="i112" sort="Set"/>
              <REL value="1" binder="m" id="i110" idref="i102" sort="Set"/>
            </APPLY>
          </target>
        </LAMBDA>
        <APPLY id="i88" sort="Prop">
          <MUTCONSTRUCT uri="cic:/Coq/Init/Peano/le.ind" noType="0"
            noConstr="1" id="i95" sort="Prop"/>
          <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"
            noConstr="1" id="i93" sort="Set"/>
        </APPLY>
        <LAMBDA sort="Prop">
          <decl id="i16" type="Set" binder="m">
            <MUTIND uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0" id="i30"/>
          </decl>
          <decl id="i32" type="Prop" binder="H">
            <APPLY id="i44" sort="Type">
              <MUTIND uri="cic:/Coq/Init/Peano/le.ind" noType="0" id="i50"/>
              <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"
                noConstr="1" id="i48" sort="Set"/>
              <REL value="1" binder="m" id="i46" idref="i16" sort="Set"/>
            </APPLY>
          </decl>
          <target>
            <APPLY id="i56" sort="Prop">
              <MUTCONSTRUCT uri="cic:/Coq/Init/Peano/le.ind" noType="0"
                noConstr="2" id="i72" sort="Prop"/>
              <MUTCONSTRUCT uri="cic:/Coq/Init/Datatypes/nat.ind" noType="0"
                noConstr="1" id="i70" sort="Set"/>
              <REL value="2" binder="m" id="i68" idref="i16" sort="Set"/>
              <REL value="1" binder="H" id="i63" idref="i32" sort="Prop"/>
            </APPLY>
          </target>
        </LAMBDA>
        <REL value="1" binder="n" id="i14" idref="i0" sort="Set"/>
      </APPLY>
    </target>
  </LAMBDA>
</ConstantBody>
```


Figure 3: The same proof in XML.

```
<?xml version="1.0"?>
<omdoc:proof
  xmlns:omdoc="http://www.mathweb.org/omdoc"
  xmlns:m="http://www.w3.org/1998/Math/MathML">
  <omdoc:derive>
    <omdoc:FMP>
      ∀n : nat.0 ≤ n
    </omdoc:FMP>
    <omdoc:method xref="Intros+LetTac">
      <omdoc:proof id="i0">
        <omdoc:declaration>
          <omdoc:presentation><omdoc:use format="pmml">n</omdoc:use></omdoc:presentation>
          <omdoc:type>
            nat
          </omdoc:type>
        </omdoc:declaration>
        <omdoc:derive>
          <omdoc:FMP>
            0 ≤ n
          </omdoc:FMP>
          <omdoc:method xref="By_induction">
            n
            λm : nat.0 ≤ m
            <omdoc:proof id="i88" style="arity: 0">
              <omdoc:label>O</omdoc:label>
              <omdoc:derive>
                <omdoc:FMP>
                  0 ≤ 0
                </omdoc:FMP>
                <omdoc:method xref="Apply">
                  (le_n.)
                </omdoc:method>
              </omdoc:derive>
            </omdoc:proof>
            <omdoc:proof id="i16" style="arity: 2">
              <omdoc:label>S</omdoc:label>
              <omdoc:declaration>
                <omdoc:presentation><omdoc:use format="pmml"><mi>m</mi></omdoc:use></omdoc:presentation>
                <omdoc:type>
                  nat
                </omdoc:type>
              </omdoc:declaration>
              <omdoc:hypothesis inductive="yes">
                <omdoc:label>H</omdoc:label>
                <omdoc:FMP>
                  0 ≤ m
                </omdoc:FMP>
              </omdoc:hypothesis>
              <omdoc:derive>
                <omdoc:FMP>
                  0 ≤ m + 1
                </omdoc:FMP>
                <omdoc:method xref="Apply">
                  (le_S . .)
                  <omdoc:premise xref="i32"><ci xref="i63">H</ci></omdoc:premise>
                </omdoc:method>
              </omdoc:derive>
            </omdoc:proof>
          </omdoc:method>
        </omdoc:derive>
      </omdoc:proof>
    </omdoc:method>
  </omdoc:derive>
</omdoc:proof>
```

Figure 4: The same proof in OMDoc.

Proof of $\forall n : nat.0 \leq n$.
Assume $n : nat$.
We proceed by induction on n to prove $0 \leq n$:
  Base case ($0 \leq 0$): immediate by le_n.
  Inductive step:
    Given $m : nat$ we need to prove $0 \leq m{+}1$ under the inductive assumption H that $0 \leq m$.
    Immediate by le_S and H.

Figure 5: The expected rendering of the proof.

It is worth noticing that the proof is now already structured in a way that resembles more closely the way mathematicians write their own proof. In particular, the thesis of the inner proofs really help in understanding the reasoning underlying the original proof. In the next year we plan to release other two deliverables (D2.e and D2.f) consisting in a set of stylesheets able to provide a reasonable human-readable representation of the proofs starting from the OMDoc documents. In particular, for our simple case we expect a rendering similar to the one provided in Fig. 5, but with the possibility of interacting with the proof (following hyperlinks to the lemmas and definitions involved, expanding or hiding trivial parts of the proof, etc.)

4 The transformation of mathematical items

Figure 6 describes the overall organization of the XSLT stylesheets in charge of transforming a mathematical item (theorems, definitions, etc.) into its intermediate representation. If the item has a user-provided natural language annotation this takes precedence over any other processing. Otherwise, the item is transformed into an OMDoc item, and we proceed in processing its microstructure. If the term to be processed has sort Prop it is recognized as a proof, and processed in a special way (producing OMDoc proof markup); otherwise (if it has sort Set or Type), it is regarded as a term, a formula, or a type, and it is directly transformed into a MathML expression.

4.1 rootcontent.xsl

The main file of the transformation phase is called `rootcontent.xsl`. It imports three stylesheets:

- `annotatedcont.xsl`
- `links_library.xsl`
- `getter.xsl`

`links_library.xsl` is a pretty complex library of functions responsible for assembling URL's from URI's, escaping the required characters. All URL's are dynamic: a typical URL is a request to some agent (an XSLT processor) to apply a pipe of stylesheets to a document retrieved by another agent (the getter, whose URL is specified in the request) via the URI of the document.

The stylesheet `getter.xsl` is a simple wrapper for the "getter", that is a web service mapping URI's to URL's.

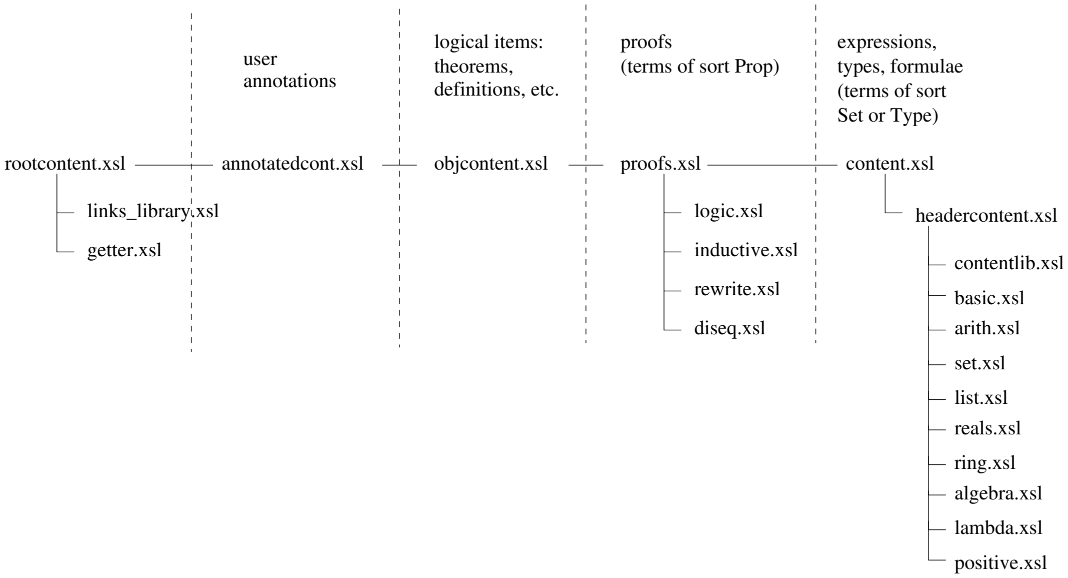

Figure 6: Inclusion structure for XSLT stylesheets to intermediate representation

The main file is annotatedcont, discussed below.

`rootcontent.xsl` is also responsible for loading and storing into variables the inner types and annotations.

4.2 annotatedcont.xsl

Proofs of the library may have user-supplied natural language annotations. If this is the case, they are directly rendered. If no annotation is present, we pass into a special mode called NOANNOT. Most of the real transformation process is done in mode NOANNOT. The file `annotatedcont.xsl` includes `objcontent.xsl`, that is essentially the root file in mode NOANNOT. The current management of annotations is still very experimental; the forthcoming task T4.3 has been explicitly devoted to this issue.

4.3 objcontent.xsl

The file `objcont.xsl` is the real root of the transformation phase. It maps mathematical items (Theorems, Definitions and so on) into OMDoc items, and then proceed in processing the microstructure (formulas, terms, proofs, etc.) in mode NOANNOT. The root of the transformation phase for the microstructure is `proofs.xsl`.

4.4 proofs.xsl

Terms of the Calculus of inductive Constructions of sort Prop are recognized as proofs. A complex set of transformation is applied in this case in order to attempt a reconstruction of the proof into natural language. If the term does not have sort Prop, it is regarded as a "pure" term (an expression, a formula, or a type), and is processed in mode PURE, by `content.xsl` (included in `proofs.xsl`). The stylesheet `proofs.xsl` also includes the following specialized stylesheets, in charge of matching special cases and constructions of the underlying logical system:

- `logic.xsl`, handling introduction and elimination rules for basic logical operators, such as *and*, *or*, *exists*, etc.
- `inductive.xsl` handling inductive proofs.
- `rewrite.xsl` handling rewriting rules.
- `diseq.xsl` handling chains of (in)equalities.

4.5 content.xsl

The last stylesheet, `content.xsl`, is in charge of transforming terms of the Calculus of Inductive Constructions (expressions, types and formulas) into the correspondent MathML expressions (we have suitably extended the core set of MathML content with a bunch of new csymbols in order to cover the missing markup). `content.xsl` includes (via `headercontent.xsl`) several stylesheets which are responsible of matching mathematical operators and transforming them into the suitable element of MathML content (or a new csymbol):

- `contentlib.xsl` Utility functions for other stylesheets.
- `basic.xsl` Basic logical operators and simple datatypes.
- `arith.xsl` Arithmetical operators.
- `set.xsl` Basic ("naive") Set Theory.
- `list.xsl` Lists.
- `reals.xsl` Real numbers.
- `ring.xsl` Rings.
- `algebra.xsl` Algebra.
- `lambda.xsl` Lambda calculus.
- `positive.xsl` "Fast integers".

All these stylesheets are currently *automatically generated* starting from a metanotation entirely developed inside MoWGLI.