The Marriage of MathML and Theorem Proving

Hanane Nacri and Laurence Rideau
INRIA Sophia Antipolis, BP. 93, 06902 Sophia Antipolis Cedex, France

Abstract: Tools dedicated to mathematics need to display formulas in a manner that is close to the typesetting practice of mathematical literature and to make them easily accessible on the Internet. This paper presents our customisation of FIGUE, an interactive two dimensional layout engine, to display mathematics and to support the MathML standard.

1 Introduction

This paper describes our use of MathML in the context of user-interfaces for theorem proving. Formal proofs on the computer are generally composed of logical formulas — the goals being proved — and commands gathered in a proof script that is interpreted by the theorem prover to break down the goals into simpler ones and build the proof of the initial goal. Any user-friendly interface for a theorem prover needs to layout mathematical formulas appearing both in statements and commands. Developing proofs on the computer is made easier by adequate mathematical notation: laying out the mathematical formulas in a manner that is very close to the typesetting practice of mathematical literature increases the readability of the mathematical formulas and speeds up the proving process in a significant manner. This is especially true for the kind of logical engine we use, where interactive proving is preferred to automatic proof search.

Once a proof is complete, dissemination is an important part of its life cycle. Proof presentation capability is then needed, for example for didactic purposes, or, in an industrial context, to allow inspection of a formal development by a third party for certification or security evaluation. Here we have the choice to communicate the compiled form of the proof (as in the HELM project http://www.cs.unibo.it/~aserti/HELM/home.html) or the script source; these two options can be compared, in the domain of programming, to the choice between distributing compiled libraries versus distributing source code. We have chosen to distribute the scripts, possibly giving a natural language representation of the commands, closer to the standard proofs built by mathematicians on the paper. In any case (script or compiled form), using mathematical notation is essential, making the proofs more easy to read and to understand.

To display proofs, we use FIGUE [NR01, NR00], an incremental interactive 2-dimensional extensible Java library developed in our research team. FIGUE was first designed to display structured data like programs, that need only an almost linear layout. To adapt FIGUE to our needs, we have first added some new graphical constructors to display mathematical formulas such as roots, integral signs, arrays, matrices, fractions, etc. For this purpose, we have followed the MathML standard to build our constructors with the same semantics and attributes.

By using MathML when extending FIGUE, we have made it very easy to translate our FIGUE layout structure (a box tree, whose nodes correspond to the FIGUE graphical constructors) to XML source including MathML-presentation markup for the mathematical formulas. This functionality makes it possible to communicate proofs (goals and scripts) on the Web, as long as one uses a MathML-compliant browser. Conversely, our tool can be used as a MathML layout engine. Our program is also suited to handle MathML-content compliant data.

This paper first describes how MathML is used as a standard to add mathematical constructors in FIGUE, and then how MathML is supported in the context of theorem proving: firstly by
2 Mathematical Formula Layout in the context of theorem proving

The underlying goal of our work is to build user-friendly interfaces and tools for theorem proving; in particular we develop pcoq [ABPR01], a graphical interface for the coq [HKPM97] theorem prover. For this purpose we need to be able to display 2-D mathematics for which we use the JAVA FIGUE library. This section presents FIGUE, and explains how we have extended the library by adding new mathematical constructors using MathML as a standard, and finally describes briefly our use of FIGUE in the context of theorem proving.

FIGUE is an independent layout module, offering the possibility to present graphically the structured objects of documents in two dimensions. Like the majority of layout tools, FIGUE is based on the idea of layout boxes [Knu90] to represent structured objects of the displayed document. FIGUE directly manipulates a box-tree which describes how to layout the objects on a page and memorises the dependence relation (the hierarchy of inclusion) between graphical objects. Each node in the tree represents a box (associated to a graphical constructor) including all its child boxes and the leaves are the tokens (basic units like strings) appearing in the final displayed document. Figure 1 shows a simple example of the box representation of a mathematical formula.

![Box representation of a mathematical formula](image)

Figure 1: (a) Mathematical formula. (b) Corresponding boxes. (c) Box-tree representation.

Each FIGUE constructor (or combinator) is written in JAVA according to a predefined interface. Each combinator has its own formatting algorithm to specify the relative position of its sons and to determine the size and the alignment of its bounding box (the whole box). This algorithm updates the box’s graphical properties (or attributes specific to a box): origin, size, and alignment, by taking into account the display area parameters like the page width and the different parameters related to the graphical context of each box (when using multiple fonts for example). Once the formatting task is done, FIGUE does a single pass to display the formatted objects of the document according to their graphical contexts (font, colour, background, coordinates) and draws the needed symbols or typographical characters (for example, the mathematical root symbol). For this purpose, each constructor needs to be able to express where lines, symbols, and sub-expressions are drawn when it is being used; it must also be able to react to messages indicating that a sub-expression has changed sized (due to editing) or that the size allocated to this combinator has changed, due to a size change in another combinator. Note that FIGUE has an incremental layout algorithm, with minimal re-drawing actions when data is modified, for a more efficient user-friendly result. FIGUE offers several default graphical constructors: Paragraph, Horizontal, Vertical, Atomic. To display mathematics we have added some new constructors such as Root, Fraction, Table, Subscript, Superscript, Subsubscript, etc., implementing the different methods dedicated to formatting,
drawing, and allowing incremental redisplay. These new constructors have been designed following MATHML-presentation recommendations.

Within the scope of proof development, we use FIGUE to lay out the proof data, including mathematical notation. In our interactive proof environment, we provide ways to edit structurally proof data (scripts made of commands and formulas) following an abstract syntax definition (formalism). A proof is represented by an abstract syntax tree (for the formulas it corresponds to the MATHML-content markup) which is transformed into a FIGUE box-tree (corresponding to the MATHML-presentation markup) to be displayed. Our approach separates the content markup from the presentation markup while keeping a natural relationship between them. The presentation tree (box-tree) is produced by applying the transformation rules written in the abstract language called PPML (Pretty Printing Meta Language) on the abstract syntax tree. For each node in the syntax tree, PPML first searches the appropriate transformation rule by pattern matching and then applies it. Figure 2 shows an example of a transformation of a mathematical formula syntax tree into a FIGUE box-tree.

![Diagram showing transformation of a formula's syntax tree into a FIGUE box-tree using PPML rules.](image)

Figure 2: Transformation of a formula’s syntax tree into a FIGUE box-tree using PPML rules.

To ensure adequate notation, end-users must be able to customize the layout of proofs with their own notation by specifying their own PPML transformation rules. For example a determinant could

![Image showing example of mathematical notation: linear algebra.](image)

Figure 3: Example of mathematical notation: linear algebra.
be displayed as a table as shown in Figure 3, while the default layout of a determinant would have the functional form:

\[(\text{det}3.3 \ a \ b \ c \ d \ e \ f \ g \ h \ i)\]

Note that, in the context of interactive proving, the data displayed on the screen can be selected with the mouse, making the problem rather different from the passive mathematical layout in \TeX. The PPML compiler maintains the correspondence between the box-tree structure and the syntax tree. This powerful interaction mechanism allows for more intelligent manipulations than a simple copy-paste, such as the development, the simplification, and the modification of the underlying mathematical expressions. This type of manipulations is essential in user interfaces. For example, the PFCQ system uses FIGUE and exploits this powerful interaction mechanism to develop a proof by selection and to manipulate algebraic formulas [BKT94, Ber97].

3 Theorem proving and MathML

Once a proof or a theory (a set of proofs) is complete, the author usually wishes to publish it. In the case of COQ developments, some tools exist that automatically generate HTML source with formulas displayed in ASCII syntax. Using these tools, the advantage of the graphical interface for mathematical notation is lost. Presently, our aim is to make proof data easily accessible and readable on the Internet, enabling to exchange and communicate this data with other users and between different scientific applications. To reach this goal, we have chosen to support MathML in our environment by generating both MathML-content and MathML-presentation formats.

3.1 MathML-presentation

As we have followed MathML-presentation recommendations to design our mathematical constructors, the correspondence between MathML-presentation elements and FIGUE constructors is natural, which makes the translation easy. For each FIGUE graphical constructor, we have defined what the corresponding MathML-presentation element is. From the displayed graphical objects, we are able to generate an XHTML document including MathML-presentation. The generated XHTML represents the structured document corresponding to the FIGUE display; it can be displayed by any browser supporting MathML (Mozilla, Amaya [AMA], etc.). Figure 4 shows how Amaya displays an XHTML document which is automatically generated from the proof shown in the example in Figure 3. This functionality makes it possible for proof developers to have automatically an electronic document version of their proofs (goals and scripts) which is easy to communicate on the Internet and to exchange with other users.

Conversely, to be able to display MathML-presentation documents, we have developed in FIGUE a module to interpret the MathML-presentation format. This module uses a DOM-based (Document Object Model) generic XML parser to analyse and validate the MathML document and produces a DOM tree representing the MathML document. Next, this DOM tree is translated into a FIGUE box tree which can be displayed. This module establishes the link between the box structure and the MathML element, allowing thus the editing of MathML documents. To translate a MathML-presentation document into a FIGUE box tree, we adopt two methods: the first method makes a direct translation, associating through some Java code each MathML element to a corresponding FIGUE constructor. The second method applies the XPPML transformation rules formalism to translate each MathML-presentation element into a FIGUE box. The XPPML \(^1\) technique offers the user the possibility to add, in a declarative way, his own transformation rules, depending on his own style and preferences for presenting the mathematical formulas.

\(^1\)XPPML is based on the PPML ideas and is a simplified form of XSLT for transforming XML documents
3.2 MathML-content

In our proof environment, the manipulated objects containing mathematical formulas are structured data, represented by abstract syntax trees allowing powerful symbolic manipulation. Using such a tree-like data structure makes translation from our proof documents to MathML-content rather easy. All logical formulas appearing in the proofs can be represented directly in MathML-content markup language while keeping the same semantics, but the other abstract objects like theorem statements or definitions do not have any MathML-content corresponding element. To bypass this problem, we have adapted our XML markup to present these abstract objects as XML tags and we only use MathML-content elements to encode the embedded mathematical formulas.
To present a whole formal proof in a standard markup language such as MathML-content, we need to be able to build new mathematical objects in this language. Presently, we are only able to translate the formula subparts of proofs into MathML-content, making it possible to collaborate with other symbolic systems using MathML as a communication protocol (for example to ask a computer algebra system for a computation).

Figure 5 shows the translation of the abstract syntax tree corresponding to the theorem declaration: 

**Theorem Not_not:** \( \forall A \in Prop \quad \neg (\neg A) \Rightarrow A \). Introduce XML code, where the pure statement part is encoded using MathML-content, with the type declaration of \( A \) represented by a condition assuming that \( A \) belongs to the right type (we are not sure that this is the right way to express that!). The non-formula nodes of the abstract tree are encoded by XML tags, such as `<theorem>` or `<ident/>`.

![XPPML transformation rule for a logical relation element.](a) XPPML transformation rule for a mathematical open interval element.](b)

Default XPPML transformation rules have been specified for each MathML-content element. As in the case of PPML, the user can specify his own style for displaying the MathML-content elements with his own notation. As a simple example, let us consider an XPPML rule used to specify the display of a mathematical formula containing both logical relations and intervals encoded in MathML-content markup language using an apply element and an interval element (see Figure 6).

We associate an apply MathML element whose first child is a relational operator `<in/>` and the two other children are \( *x \) and \( *y \), with a Row graphic constructor having three elements. In Row, the first and last elements are the results of recursive calls of the display function on \( *x \) and \( *y \), respectively, while the second element is an Atom box whose value is the result of an external Java function call, `getUnicodepp`, returning the unicode character corresponding to relational operator op (when the op's value is in, this function returns the unicode character \('u2208\)'). In the following, for a better readability, we write the transformation rules with the high level language PPML. In PPML, the rules corresponding to the transformation, as discussed in the previous example (Figure 6), are written as follows:
apply($op=in, *x, *y$) → $<$Row$>$ $*x$ getUnicodepp($op$) $*y$$>$

$*z$ as interval($*x, *y$) → if isOpenIntervalpp($*x$)
then
$<$Row$>$ "$\ldots$$" $<$Row$>$ $*x$ "$\ldots$$" $*y$ "$\ldots$$"$$>$
end

As an example, Figure 7 shows the encoding of the mathematical formula $x \in [a, b]$ in MathML-
content markup and the corresponding transformation rule into a FIGUE box-tree.

![XPPML Transformation](image)

**Content Markup**

```xml
<apply>
  <in/>
  <ci>x</ci>
  < interval_closure="open"> 
  <ci>a</ci>
  <ci>b</ci>
  </ interval_closure>
</apply>
```

![FIGUE box-tree](image)

Figure 7: The transformation of the mathematical formula $x \in [a, b]$ from MathML-presentation
markup to a FIGUE box structure.

The XPPML transformation rules give the user a powerful way to specify his own notation. Some
mathematical objects change notation depending on the context (e.g. the country). For instance,
in the previous example, instead of writing the open interval $[a, b]$, one may want to write it $(a, b)$.
This will be performed by the following XPPML rule:

$*z$ as interval($*x, *y$) → if isOpenIntervalpp($*x$)
then
$<$Row$>$ "$\ldots$$" $<$Row$>$ $*x$ "$\ldots$$" $*y$ "$\ldots$$"$$>$
end

4 Conclusions

This paper has described the collaboration of theorem proving and MathML, which is concretely
available in PCoq in the following ways:

- Using MathML as a standard, we get the right basic constructors, with the right attributes
  in our layout engine.
- The similarity of MathML-presentation and FIGUE constructors makes the translations be-
  tween both formalisms easy, thus enabling the display of proofs on the Internet.
- Using a tree-like data structure makes the translation to MathML-content easy, thus making
  it possible to collaborate with other symbolic systems with MathML as a communication
  protocol.

PCoq and FIGUE are available at:
http://www-sop.inria.fr/lemme/pcoq and http://www-sop.inria.fr/croap/figue
References


