Guided Research Final Report

Coping with Content Representations of Mathematics in Editor Environments: nOMDoc mode

Darko Pesikan <d.pesikan@jacobs-university.de>
Jacobs University Bremen, prof. Michael Kohlhase

Spring 2007

Abstract

The amount of mathematical knowledge available in the electronic format is enormously increasing. This creates a need for many new technologies which must better support the interaction between humans and this vast information pool. Especially promising direction is content-based ways of representing mathematics, as this establishes grounds to real automation, interoperability, transformation and processing of mathematics. However, the main problem of semantic-based techniques is the actual authoring cost of semantically enriched documents [AA04]. As an example how can management of such documents be greatly facilitated, we present nOMDoc mode, an extension of EMACS editor which exploits specific traits of mathematical documents written in an XML-based format that embraces both Content MathML and OpenMath: OMDoc format. In particular, the mode features a direct semantic math web search from the editor and further explores possible interactions between a search engine and an ccc editor environment.

Contents

1 Introduction and motivation

1.1 OMDoc and Content-Based Representations

The OMDoc [Koh06] (Open Mathematical Documents) format is an open markup language for mathematical documents and the knowledge encapsulated in them. As other mathematical markup schemes, OMDoc is an XML application. The format encapsulates both OpenMath [BCC+04] and Content/Presentation MathML [ABC+03] formats, and thus enables and motivates content-based representation of mathematics. Since it supports different mathematical formats and provides infrastructure for markup of large-scale mathematical objects such as theories, the OMDoc format serves us as an excellent subject format for demonstrating how to handle peculiarities in editing or otherwise dealing with content-oriented representations of mathematics.

Content-oriented representations raise a number of pragmatic issues when combined with standards they should abide by such as XML, but despite that they are an extremely powerful and natural approach. Its rationale is in a famous statement by John R. Pierce that mathematics and its presentation should not be viewed as one and the same thing [Pie80]. Namely, if instead on focusing on contextually volatile, often multifaceted particular rendering of a mathematical expression, we decide to encode underlying mathematical structure of an expression without regard to how it is presented aurally or visually, we establish a sound infrastructure for communication between different software systems, which is virtually impossible with presentation marked-up mathematics. That's why context-oriented respresentations mean moving towards real formalizations of the mathematical content.

The current state of the art in editing and searching for content representations is not overdeveloped. *Hermes* [1] system is one remarkable attempt in this direction. The system helps the content oriented LaTeXauthoring for scientific documents supporting manual and automatic generation of MathML content, enabling the author to add and recover semantic depth and clarity of documents written in LaTeX. However, text editors do not stand up well in this respect given the potential some of them contain - emacs in the first place - and that is the prime reason why we have embarked upon this investigation. In it we aim to help creators of content-oriented markup documents in OMDoc format be more efficient, improve documents' content and in turn make mathematical knowledge more accurate, more connected and less redundant.

1.2 Emacs

emacs is a class of highly customizable text editors [2] with collosal set of features. It is one of the most ported computer programs of all times, running on a variety of operating

[1] http://hermes.roua.org

[2] In contrast to *word processors* which give more emphasis on formatting text than on editing plain text (though this is not to say that emacs can't format text too).

systems, including most Unix-like systems (GNU/Linux, the various BSDs, Solaris, AIX, IRIX, Mac OS X), Microsoft Windows and OpenVMS. Many emacsen [3] have appeared, but the most commonly used is GNU Emacs [4], which we focus on here.

EMACS' most curious feature is that it has embedded a whole programming language in itself, a dialect of Lisp called *Emacs Lisp*. In Emacs Lisp, variables and entire functions can be modified on the fly, without having to recompile or restart the editor. As a result, the behavior of the editor can be modified greatly, either directly by the user, or by loading bodies of Lisp code. This feature allows the editor to support writing of virtually all kinds of documents, and some versions even have MP3 music player, computer games or a web browser integrated.

In particular, content sensitive *major modes* for a wide variety of file types can also be programmed in Emacs Lisp and are of our special interest. A major mode is a set of variables and functions used to tune the EMACS editor for editing a particular sort of text in it. Major modes are mutually exclusive and each buffer [5] has one major mode at any time. This is in contrast to minor modes, which can be concurrently active in just one buffer. Major modes are typically used for highlighting, indentation, compiling and similar tasks related to the document types they are designed for. They can be created either from scratch or by deriving from already existing modes: a technique remotely related to inheritance in object oriented programming and the one we employ.

Given EMACS' extensibility and portability, it clearly presents a good candidate for the environment in which one could demonstrate how to efficiently create mathematical content-oriented documents and get along well with lot of XML that is often hard to read and edit.

1.3 nXML mode and OMDoc mode

Naturally, major modes designed for editing XML documents already exist. A prominent example is the popular nXML mode [6], an evolved version of the standard XML mode. The mode's most valuable feature is validation of XML documents given the appropriate RELAXNG [Com] schema. Technically, RELAXNG schemata, similarly to DTDs, provide a description of how can valid documents of a given XML application be distinguished from a vast number of other invalid XML documents. The schemata are used in nXML mode for:

- real-time validation of the document by a clever mechanism of background re-parsing and re-validating and automatic highlighting of validity errors

[3] Plural of emacs, an analogy to ox - oxen.
[4] http://www.gnu.org/software/emacs/
[5] In EMACS' idiosyncratic terminology, a buffer is an opened file.
[6] http://www.thaiopensource.com/nxml-mode/

- completion mechanism which includes assisting in entering appropriate element names, attribute names or data values given its context.

However, although very deft, nXML mode is not especially suited for editing mathematics written in OpenMath or Content/Presentation MathML formats. In addition, nXML mode sees OMDoc only as yet another application of XML: features such as semantic sanity check and well-connectedness of the OMDoc theories obviously do not exist.

The first attempt to create an emacs major mode specifically for OMDoc format was done by Peter Jansen at Carnegie Mellon University in 2002 in form of OMDoc mode [7]. This project featured a couple of OMDoc specific options such as highlighting groups of tags with different colors according to their common semantics. However, the mode did not take into account other semantic properties of OMDoc documents, nor did it ripe benefits of the already existing nXML mode. Ultimately the project was abandoned.

2 nOMDoc mode: the new OMDoc mode

2.1 Overview

In light of the abovedescribed current state of affairs and the need to attempt to balance the power of content representations with easiness of its editing, we present nOMDoc major mode - a new OMDoc mode for emacs text editor.

The nOMDoc mode was derived from nXML mode, which in practice means it contains all its features and more. As OMDoc format was built on the grounds of XML, it is indeed a sound idea to reflect this relationship in emacs modes too. In that way we follow an already existing standard and potential nOMDoc mode users who are already familiar with nXML mode can have a smooth transition to nOMDoc mode. The OMDoc format specification contains a RelaxNG schema [8] which works well when employed by nXML mode. Since nXML mode is well documented, we shall focus on the specific features of nOMDoc mode.

Some of the mode's features were inspired by ideas in old OMDoc mode. The old OMDoc features that we kept and improved include and are limited to highlighting, templates, and the timestamp routine. Indentation, completion, navigation and validation routines have not been included as those tasks are better supported by nXML mode itself.

[7] Check https://svn.omdoc.org/repos/omdoc/trunk/lib/emacs/omdocmode
[8] https://svn.omdoc.org/repos/omdoc/rnc/omdoc.rnc

Function name	Keyboard Shortcut
nomdoc-om-completion	CTRL+c c
nomdoc-sanity-check-theory	CTRL+c k
nomdoc-sanity-check-openmath	CTRL+c m
nomdoc-sanity-check-buffer	CTRL+c f
nomdoc-build-registry	CTRL+c b
nomdoc-update-registry	CTRL+c p
nomdoc-generate-om	CTRL+c g
nomdoc-math-outline	CTRL+c o
nomdoc-math-show-om	CTRL+c w
nomdoc-roll-element	CTRL+c r
nomdoc-unroll-element	CTRL+c u
nomdoc-insert-time-and-user-at-point	CTRL+c t
nomdoc-insert-new-buffer-strings	CTRL+c n
nomdoc-math-search-string	CTRL+c s
nomdoc-math-search-xmlq	CTRL+c x
Command mode	
sanity-check-execute	x
sanity-check-ignore	i
sanity-check-execute-all	a
sanity-check-quit	q
Math search mode	
math-search-click-link	RETURN
math-search-click-next	n
math-search-click-previous	p
math-search-click-insert-xml	i
math-search-again	a
math-search-quit	q

Table 1. Interactive nOMDoc functions and their keyboard shortcuts

Not including those from nXML mode, nOMDoc mode features can be divided into seven categories: Highlighting, Templates, Semantic sanity check, Integrated math web search, Semantic completion, Math prefix notation, and Miscellaneous. As we shall see, nOMDoc mode currently focuses on OpenMath format which is a particularly appealing format for representing mathematics. This is since the format is content-oriented, very extensible and fits well in OMDoc format. The mode contains two submodes (technically just another two major modes): `math-search-mode` for displaying the found results in a separate buffer and providing navigation, insertion and conversion for them, and `command-mode` for deciding

what actions to execute given a list of items (useful for semantic integrity checks and possibly for other features in the future).

Table 1 lists all interactive functions that are specific to nOMDoc mode, and their keyboard shortcuts. There are over 30 customizable variables for the mode in a variable group `nomdoc`. Some commands are also accessible through OMDoc menu which has expert and novice modes.

We shall now give an overview of all main features.

2.2 Highlighting

The nXML mode does not provide an option to color different kind of tags with different colors. Since OMDoc specification provides different categories of markup, it makes sense to have diverse highlight too. For instance, it is useful to be able to quickly spot chunks of XML Math or CDATA embedded inside the document which is easier in math tags are highlighted with different color.

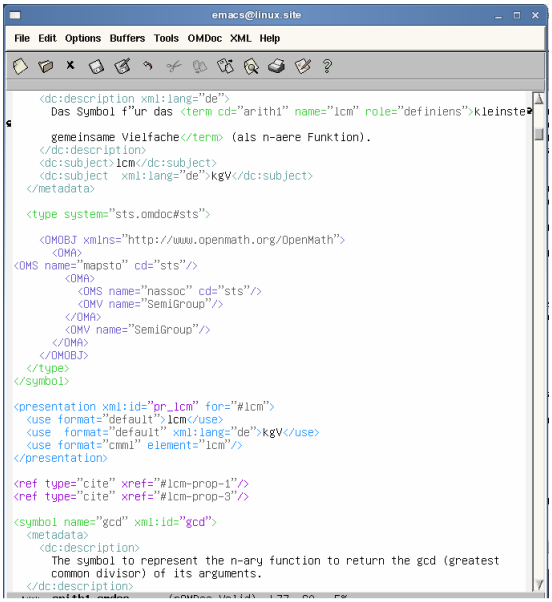

Figure 1: Default highlighting of a typical OMDoc document

The nOMDoc mode divides all OMDoc tags into a few groups such as structure keywords, math keywords and presentation keywords. It is possible to customize the color assigned to each group as well as specific tags that belong to it.

Note that the mode highlights text provided that the original nXML mode's is turned off: a simple overdraw of existing highlighting didn't work since nXML uses a non-standard way to highlight text. To turn the nXML mode's highlighting off, set `Nxml Syntax Highlighting Flag` variable in customizable group `nxml` to nil.

2.3 Templates

The nXML mode itself has good context-sensitive completion mechanisms which are driven by riping out the most of a given RelaxNG schema. A key combination `CTRL+RETURN` pressed while inputting an element or an attribute name completes the name fully given the document's context or provides a special buffer with possible completions. So-called electric slash feature provides automatic insertion of the appropriate closing tag once a string `</` has been detected.

In parallel with that, the mode has its own templates (skeletons of often used patterns of text) for all OMDoc 1.2 elements. Key combinations of form `CTRL+C CTRL+`x y, where y is a letter corresponding to an element name from an OMDoc 1.2 module associated with letter x, insert element skeletons tailored [9] to reflect often used attributes and visual layout. Once the user starts inputting a tag name, the corresponding skeleton can be completed using `META+CTRL+c`[10]. Hot points inside skeletons where text is to be inserted have been defined and the user can go back and forth through them using `META+CTRL+b` and `META+CTRL+f`.

This feature has been built using the infrastructure provided by standard library `tempo.el`, similarly as in old OMDoc mode. All skeletons are distributed into a system of menus and submenus inside emacs menu *OMDoc*.

2.4 Semantic sanity check

OMDoc provides element `theory` which helps structure and organize mathematical knowledge. Those modules of knowledge are not self-contained - often theories use symbols, axioms, definitions or theorems defined elsewhere. As a simple example, theory A may contain a definition of a symbol for division, and it happens that multiplication symbol was defined in theory B. However, multiplication is semantically necessary for defining division in our scenario and OMDoc provides an infrastructure to express this fact syntaxically. This is done via element `imports` which is used to build that framework of semantic references. Once theory B has been *imported* in theory A, objects that are in its scope (either directly or indirectly) become available in theory A too. Obviously, maintaining a large number of OMDoc documents while keeping semantically sound connections in this DAG of theories is not easy, but is instead semiautomatizable.

The nOMDoc mode tries to help there by creating those inheritance relationships or suggesting modifications to already existing `imports`. It does this by inspecting theory's content and analysing the usage of foreign symbols not defined in it. At this stage, nOMDoc mode understands theories only as OpenMath Content Dictionaries of symbols that can be used as `OMS` elements.

[9] It appears that we can tailor even skeletons in emacs :)

[10] In Emacs vernacular, META means Alt key on modern PCs.

For efficiency purposes, all current links between theories and names of all symbols defined in them (given a specified repository of OMDoc files) are cashed in a registry of symbols. The registry follows XML standard and is updated automatically upon saving an OMDoc document or on user request. The RelaxNG compact syntax for it is as follows.

```
element nomdoc-registry {
  element theory {
    attribute path { text },
    element symbol {
      attribute name { text },
    }*,
    element imports {
      attribute from { text }
    }*
  }*
}
```

Currently, the grammar and functions that use the registry are not aware of subtheories inside a theory, and behavior of the mode in this case in unpredictable. Further development will include axioms, definitions and assertions in the registry, because OMDoc supports linking them with proofs by `premise` elements.

We can use this registry for a number of purposes, one being to compute an *optimal* set of dependencies for a given theory as described above. What is meant by optimal here is not trivial, and we confine ourselves to our own inductive definition.

Definition 1.

- An empty set is an optimal theory dependency DAG.
- A set S of theories is an optimal set of dependencies for a theory $t \in T$ (T being the set of all theories) given an optimal theory dependency DAG G which does not contain S, a set U of mathematical objects used by t and a function $g : T \mapsto \mathcal{P}(O)$ (where O is the set of all mathematical objects) such that $g(t_i)$ is a set of all objects theory t_i defines, iff it satisfies the following four conditions:
  1. $t \notin S$, self dependency is not allowed
  2. $\forall o \in U : \exists m \in S : o \in g(m)$, we want only sufficient sets of theories
  3. $\forall m \in S : f(m) \cap U \neq \emptyset$, we want only 'necessary' sets of theories (suspending our awareness of dependencies until 4.)
  4. for no two theories in S there is a path in G from one theory to another (for if it is so, by inheritance only choose the source theory of that path)

Intuitively, the scope of a certain set of dependencies can be thought of as a mountain chain in DAG with the optimal set being all the highest peaks in that massif.

In practice, we might not have an opportunity to build imports elements inductively starting from the most basic theories as suggested by the definition. Nevertheless, given the algorithm below, we can improve over time even when initially non-optimal dependencies are given. This is because once optimal sets become part of the DAG, this optimality *spreads* over time to other nodes since optimality is dominant. A sketch of the algorithm employed by nOMDoc mode follows.

Algorithm 1 FIND_OPTIMAL_DEPENDENCIES(*active_buffer*, *registry*)

```
S ← ∅
G ← dependency graph constructed from registry
T ← values of cd attributes of all OMOBJ elements in active_buffer
while T ≠ ∅ do
  choose a theory t ∈ T and remove it from T
  compute all descendants of t by BFS-ing G and put them in D
  for all d ∈ D do
    if d ∈ T then
      remove d from T
    end if
    if d ∈ S then
      remove d from S
    end if
  end for
  S ← S ∪ t
end while
return S
```

Finding optimal sets of dependencies is implemented in nOMDoc mode through functions `nomdoc-sanity-check-theory` and `nomdoc-sanity-check-buffer`. The first function checks only the theory in which the point is, while the second one checks the whole buffer. The functions return temporary buffers with a list of suggestions for improvements of `imports` elements if any (adding, removing or substituting existing imports) with justifications for doing so. A typical example is: `Theory X: add Y (it offers e.g. symbol z)`.

The registry can also be used to check validity of `OMS` elements and/or completeness of existing theories with respect to their usage as Content Dictionaries. This is done by function `nomdoc-sanity-check-openmath`. After parsing the active buffer, the function lists all warnings when the following situations are encountered:

- There is more than one theory in the repository with the same name as the value of `cd` attribute in some `OMS` element

- There is no theory in the repository with the same name as the value of `cd` attribute in some `OMS` element
- The symbol with the same name as the value of `name` attribute in some `OMS` element is not defined in any theory

2.5 Semantic completion

Another usage of the registry is in semantic completion of various attribute values. Here we present completion of `OMS` elements from OpenMath. Using `imports` elements of the enclosing theory to compute the semantic context in which the editing piece of OpenMath resides, the nOMDoc mode helps in filling the values of `cd` and `name` attributes by suggesting completions solely from the list of all theories and symbols that are visible at that point. Once a `cd` attribude has been completed, the symbol space shrinks to the coresponding theory and suggests only relevant symbol completions. The potential completions are displayed in a separate buffer and are applied only on request (`CTRL+c c`).

Given a richer registry, this mechanism can be extended to other elements as well. Those include `example`, `definition` and `presentation` elements whose attribute `for` can be autocompleted using the information about what is visible in their context.

2.6 Math prefix notation

Autocompletions and templates help in creating OpenMath XML by predicting what the user wishes to write given the current context, but XML is *inherently not a succinct notation*. The way math XML code is conventionally displayed on the screen (literally or as any collapsable tree-like structure) impedes most people from figuring out the semantic content of it instantly. nOMDoc mode attempts to sooth this problem by enabling the users to input and view function applications, variables and symbols of OpenMath in a different syntax. We provide an alternative parallel S-expressions-like syntax hidden in processing tags of XML which gives a user-friendlier display of math and translated automatically below it into a corresponding OpenMath expression.

S-expressions [11] can either be single objects such as numbers or variables, or *lists* of other S-expressions. They are used as parenthesized prefix notation (Polish notation) in some programming languages. Incidentally, Lisp is one of them and we shall mimick its syntax. If an S-expression is a single object, we translate it as a constant, variable or symbol in OpenMath. If it is a list, we translate it as a function application whereby the first element in the list is the function itself and the remaining elements are its arguments. For instance, consider a piece of ultratypical example of factorial function as an S-expression:

```
(* x (fact (- x 1)))
```

[11] S stands for symbolic.

This corresponds to $x * fact(x-1)$, and the same expression in OpenMath could translate like this:

```
<OMA>
 <OMS cd="arithmetics" name="times"/>
 <OMV name="x"/>
 <OMA>
  <OMS cd="arithmetics" name="factorial"/>
  <OMA>
    <OMS cd="arithmetics" name="minus"/>
    <OMV name="x"/>
    <OMI>1</OMI>
  </OMA>
 </OMA>
</OMA>
```

In order not to influence any existing OMDoc markup or confuse XML parsers, we decided to somewhat misuse processing tags: S-expressions are entered inside them as in `<?prefix S-EXPRESSION?>`. Once a parenthesized expression has been entered, the mode generates the corresponding OpenMath element and inserts it below on `CTRL+C g`, adding appropriate *cd* attributes if there is exactly one theory in the registry that defines for each constituent symbol that plays the role of a function application.

In light of this, nOMDoc mode features two possible ways of viewing documents: `ShowAll` displays the document literally including S-expressions and OpenMath, and `MathOutline` shows only S-expressions and hides OpenMath elements that have an S-expression in front. To do this, `invisible` property in LISP of corresponding pieces of text was set to `t`. Semantic completions are also available through the same functions as when completing OpenMath (see section 2.5).

2.7 Math web search

The World Wide Web has become an greatest single resource of mathematical knowledge. This vastness of information that everybody has at their disposal creates a growing need to find the relevant information quickly. An ambitious approach to satisfy this need is through web search that is based on the *mathematical content* of information as opposed to its textual or any other representation. Such semantic search engines can search for all documents containing e.g. any definite integral from 0 to 2π, regardless of the actual name of the dummy variable in integrals. One attempt in this direction is the MathWebSearch (MWS) engine [KŞ06] developed at Jacobs University Bremen and we will focus on it here.

The MathWebSearch server [12] accepts queries in two kinds of syntax: XMLQuery and

[12] A web interface for it can be found at http://search.mathweb.org/

internal string representation. XMLQuery is essentially OpenMath/MathML decorated with some additional attributes that make the search more precise (e.g. boolean operators), and string representation is an undocumented internal format used by the engine which resembles a flattened version of the corresponding MathML expression. It currently indexes the Connexions project modules [13] but other sites are in plan too.

There seems to have been no attempts to integrate such engines into environments that are traditionally used for creating mathematical documents. In particular no EMACS mode has such a feature. In fact, the current state of the art brings the following optimal scenario. A scientist writes a research paper using his favorite editor EMACS. They need a formula they only partially remember so they switch to an Internet browser window with Math Web Search engine and copy and paste related parts of their paper (if any) to make a search query. Assuming the formula was found, they either write it from scratch in EMACS or in the best case examine the source code of the found web page and copy a corresponding piece of say MathML to the editor.

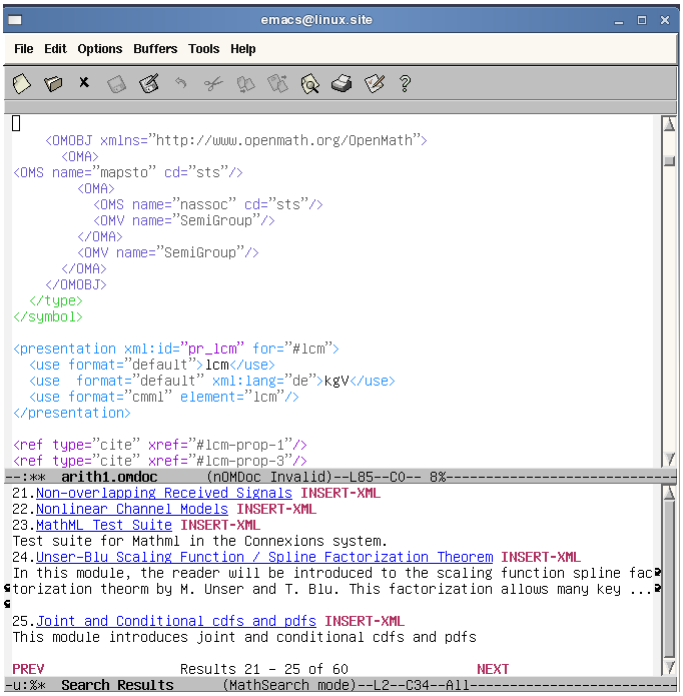

Figure 2: A web search for all indefinite integrals directly from EMACS editor

This situation is streamlined by the nOMDoc mode which offers a direct search for mathematical formulae from the editor through functions `nomdoc-math-search-string` and `nomdoc-math-search-xmlq` for the corrresponding formats of input. While string representations are inserted in the minibuffer, XML queries are edited in a separate buffer.

[13] http://cnx.org

To make use of the content of OMDoc documents, this buffer is initialized to the current active region of the editing document.

Search results are fed in a separate buffer with its own major mode, appropriate buttons and links. The whole interface somewhat resembles that of a web search engine. Results are displayed in pages which the user nagivate through using buttons `PREVIOUS` (or press `p`) and `NEXT` (or `n`). Each result is displayed as a titled link, a short description of the web-page and a button `INSERT-XML`. Upon clicking a link, a web browser window with the corresponding page opens. The `INSERT-XML` button enables the user to insert the XML of the entire formula whose subterm matched with the input query in the currently edited OMDoc document at point. Regardless of the original math XML format on the found web-page, it is possible to insert either Content MathML or OpenMath in the main document. A possible conversion from one format to another was done using appropriate XSLT stylesheets [14] and `xsltproc`.

To send the query and fetch the results a TCP-IP connection with MathWebSearch server was established (Lisp function `open-network-stream`). The communication with the server was establishing through its API described at http://search.mathweb.org/api.html.

2.8 Miscellaneous

Let us shortly review other features of nOMDoc mode that are not part of any group above.

- The current `omdoc` element in an OMDoc document [15] can be displayed in either one line, together with all its children or regularly, in possibly more than one line and with proper indentation. The former is harder to read but succinct and useful e.g. for short `OMA` constructs. The corresponding functions are `nomdoc-roll-element` and `nomdoc-unroll-element`.

- Entering some metadata sections of OMDoc mathematical objects can be automatized. Function `nomdoc-insert-time-and-user-at-point` inserts relevant `data` and `user` tags at point. For instance,
  `<dc:date action="updated" who="dpesikan">2007-05-07T22:32:59</dc:date>`
  `<dc:creator id="dpesikan">Darko Pesikan</dc:creator>`.

- `data` elements that store the last updated time in metadata of the main `omdoc` element are refreshed automatically upon saving the document.

- When a new OMDoc file is created, it is automatically filled with a default OMDoc skeleton. This includes the default XML header, `DOCTYPE` information, main `omdoc` element with an ID and five namespaces (for OMDoc itself, Dublin Core, Creative

[14] http://www.orcca.on.ca/MathML/software.html
[15] That is to say, the one in which point is located.

Commons, OpenMath and MathML) and default metadata including date, title, type, creator and rights.

3 What's next?

The first release of nOMDoc mode is already available [16] under GNU licence. The package consists of 8 Emacs Lisp files with around 3000 lines of mainly well commented code, 4 XSLT stylesheets for conversions from Content MathML to OpenMath and back and documentation. Note that documentation for each interactive Lisp function of the mode can be accessed through `CTRL+H f` from emacs. The entire set of currently available functions is displayed in Table 1.

The nOMDoc mode project is open to (but ready for) many extensions and improvements. Below we list some of those that occupied the mind of its programmer but did not turn into real code.

- Many currently available features apply only to OpenMath. However, the OMDoc format supports Content MathML too and some of the nOMDoc features for OpenMath such as math prefix notation could be extended to other formats too.
- Support for LaTeX when entering search queries would be excellent.
- Currently, we do not support expanding or collapsing of math web search results. At this point, each group of results that share the same web-page is displayed as only one link in the buffer. This could be extended (Math Web Search server's API commands `ELINK` and `CLINK` exist for that), but it is dubious whether such an interface is appropriare for a small buffer search results must occupy.
- Math prefix notation could be extended to cover OpenMath elements other then `OMV`, `OMS`, `OMI`, and `OMA` only.
- As mentioned before, symbol registry could store more OMDoc elements such as `example` or `definition` to extend scope of the current semantic completion mechanism. This may require more research to gain deeper insight about common structure of OMDoc documents to make use of knowing certain common patterns or arrangements of elements in it.

Obviously, feedback from potential users of nOMDoc mode is crucial here. Depending on it, more development in appropriate directions will happen and hopefully the mode shall continue to evolve. The most important kind of feedback we need is whether and if yes how much the authoring cost of OMDoc documents is decreased by the usage of nOMDoc mode. This is the most important variable in measuring how successful we have been.

[16] At https://svn.omdoc.org/repos/omdoc/trunk/lib/emacs/nomdocmode

4 Conclusion

We have presented the nOMDoc mode: an EMACS major mode for OMDoc documents. It demonstrates how it is possible to help in authoring content-oriented representations of mathematics.

There is a subtle notion of semiautomatizability associated with nOMDoc mode and similar modes. When writing semantically enriched mathematical documents, many tasks can be *somewhat* (sufficiently well) automated and many sequences of events can be *somewhat* predicted, but few can work flawlessly or silently, or without need for correction or other postprocessing tasks done by humans. There seems to be a special interplay between well defined syntax and elusive semantics going on when editing such documents. This gives rise to semiautomatizability of various tasks related to editing. In our opinion, this seems to be an intrinsic property of editing rather than just another surmountable technical difficulty.

All in all, the authoring step is extremely significant in the ultimate goal of reaching web-based content-oriented mathematics. After all, to process mathematics in any way, content mathematics has to be created first. Most of the mathematics on the Internet is currently still in presentation MathML format and to promote content representation of mathematics appropriate tools must be available. Demonstrating that the creation od content math doesn't necessarily have to be tedious and discouraging is an important step in the mission towards web-based mathematics and speeding up the Buchberger's Mathematical Creativity Spiral. Hopefully, the nOMDoc mode gives some insight and exemplifies a range of techniques that can be employed for this purpose.

References

[AA04] Stefano Zacchiroli Andrea Asperti. In *Searching Mathematics on the Web: State of the Art and Future Developments*. AMS/SMM special Session, Houston, May 2005, 2004.

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3c recommendation, World Wide Web Consortium, October 2003. Available at http://www.w3.org/TR/MathML2.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. `http://www.openmath.org/standard/om20`.

[Com] RELAX NG Technical Committeee. Technical report, International Standard (ISO/IEC 19757-2) by ISO/IEC JTC1/SC34/WG1.

[Koh06] Michael Kohlhase. In *OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]: Foreword by Alan Bundy (Lecture Notes in Computer Science)*, Secaucus, NJ, USA, 2006. Springer-Verlag New York, Inc.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[Pie80] John R. Pierce. In *An Introduction to Information Theory. Symbols, Signals and Noise.* Dover Publications Inc., 1980.