Reasoning about Theory Morphisms

Guided Research : Final Report

Alin Iacob
Supervisors: Prof. Dr. Michael Kohlhase, Dr. Florian Rabe

Jacobs University

May 2009

Abstract

The MMT module system for mathematical theories provides a fresh alternative for organizing mathematical data. MMT builds on concepts from existing proof development and mathematical knowledge management systems and organizes knowledge hierarchically in a theory graph, where nodes represent theories and edges are theory morphisms. A shortcoming of MMT is that one cannot quantify over theory morphisms, nor express relations between them, as required by complex mathematical texts. We attempt to rectify this and at the same time extend existing work on functor representation. We also provide a way to translation the new concepts into the old syntax, while retaining its power.

1 Introduction

The volume of mathematical knowledge is estimated to double every ten to fifteen years [Odl94], faster than our ability to organize it in a meaningful way. The traditional - and currently the most pervasive - practice of dealing with this knowledge is peer-to-peer communication (either direct or via articles), using informal language. A mathematician is thus able to mentally store complex theories and relationships and to successfully understand a new text which builds on already known concepts. However, even an able mathematician is bounded by memory constraints (limiting the number of concepts in use at one time), and by the number of abstraction levels she can use simultaneously. A book like Algebra I by N. Bourbaki [Bou89] can organize information in tens of such levels, with plenty of links between pieces of knowledge, even from different levels.

The advent of computers has brought with it the potential to systematically manage large data sets. This is already the case in the field of formal methods in software engineering (FMSE), where computer-aided proofs are backed up by large formal knowledge databases. The effectivity of managing this knowledge is thus vital for success in this field. However, in many current proof development systems, such as HOL [Gor88] or Isabelle [Pau94], it is tedious to represent large bodies of knowledge and the process requires a steep learning curve. Moreover, some of them are confined to a specific logical foundation (e.g. simple type theory in Isabelle), which hampers the exchange of information between systems.

With the Internet, it became clear that knowledge can be distributed in an easier way. The field of mathematical knowledge management (MKM) emerged, with the mission to capture both syntax and semantics of Mathematics in a format easy to be readable by both a browser and humans. XML-based OpenMath [BCC+04] groups mathematical symbols in publicly available content dictionaries (CDs), where they are assigned definitions. MathML [ABC+03] is able to express mathematical formulas as structured trees, with OpenMath constants and variables as leaves and operators as internal nodes. OMDoc [Koh08] goes one step further by representing statements (theorems, proofs, definitions) and theories, while keeping the XML syntax and allowing natural language to appear inside documents, when a formal syntax is not desired.

The MMT module system for mathematical theories [RK08] acts as a bridge between MKM and FMSE. It extends the OMDOC specification with views (expressing semantic relations between theories) and imports (instantiation of a theory in another); theory morphisms are formed by composing views and imports. It is XML-based, enabling web-scalability and code reuse and provides services such as semantic search. Just like OMDOC , MMT documents may include informal statements, while providing graceful degradation of services built on its structure. Following the "little theories" approach widely employed by Bourbaki [Bou89], MMT organizes knowledge in a theory graph, where nodes represent theories and edges are theory morphisms. To simplify the syntax, MMT employs the Curry-Howard correspondence [CF58, How80] for representing axioms as types and proofs as terms.

This thesis was inspired by the need to represent in MMT several new kinds of Mathematical concepts outlined by *Algebra I* [Bou89]. Our goal is to extend MMT in order to allow quantification and expressions over morphisms, typed variables and the parametrization of views.

Section 2 gives an overview on MMT and then summarizes existing results in functor representation, a concept which we extend to parametric views. Section 3 discusses the major problems that led to this research. We establish a syntax for the new MMT in Section 4.1 , followed by an intuitive explanation of the implementation (Section 4.2). Then, Section 4.3 points a way to transform the new language into the original, while preserving semantics. Finally, we sketch several judgmenent rules for verifying correctness of an MMT document in Appendix A .

2 Background

The main resource for understanding the MMT language is [RK08]. We will briefly summarize a newer version [Rab09], pointing out the main MMT concepts.

The MMT module system for mathematical theories structures knowledge into four levels of abstraction: theory graph, module, symbol and object level.

1. **Theory graph**: a collection of modules.

2. **Module**: declares either a theory (list of symbol declarations) or a view (list of assignments to symbols).

3. **Symbol**: either a constant declaration (declaring constants, types, axioms, lemmas, definitions) or a structure declaration (which declares an import from another theory). By declaring an import, all the symbols from the "imported" theory become available. In the theory graph, an import is represented by an arrow from the "imported" theory to the "importing" one.

   MMT does not have a special syntax for axioms and proofs, expressing them as constant declarations, according to the Curry-Howard correspondence [CF58, How80]. Thus, "axiom F" is represented by a constant of type "true F", and "theorem T with proof P" is represented by a constant of type "true T", with definiens "P".

4. **Object**: atomic objects (constant name, variable, structure name, morphism or view name, the special objects $\perp$ and $\top$) and composed objects, made of an operator applied to other atomic and composed objects (applications, bindings, attributions, morphism applications).

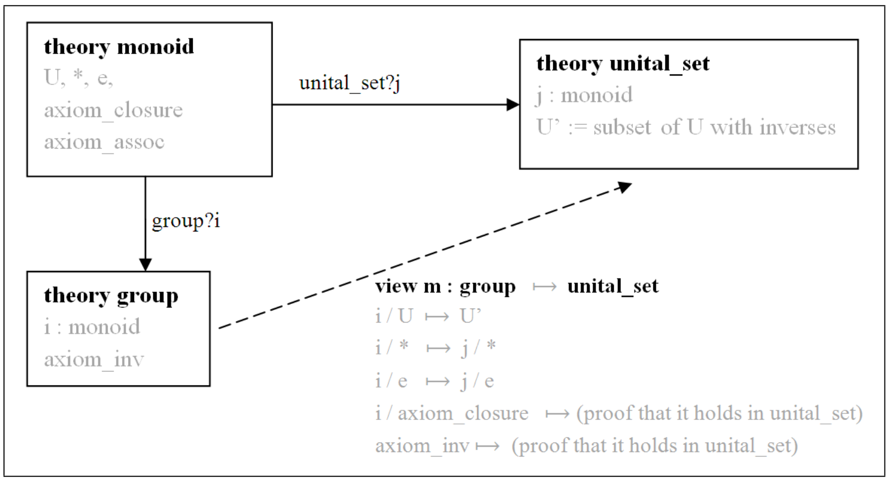
All collections of items mentioned above are ordered. This means, for instance, that a document containing modules M1 and M2 in this order is semantically different from a document containing M2 and M1. In fact, the language declared rules of well-formedness, under which the first of the above documents may be valid and the second may be not.

The complete specification, with rules for querying a library, operations on MMT expressions and an inference system defining well-formedness are found in [RK08] (the original version of the language was created as part of [Rab08]).

We illustrate the structure of this language with an example inspired from Chapter 1 of *Algebra I* by Bourbaki [Bou89].

Module level: Let us consider the theories of **group**, **monoid** and **unital set** of a monoid and one view (**m**), which shows that the unital set of a monoid is a group. Views have a theorem flavor: it provides assignments to all the symbols of S.

The associated graph is shown in Figure 1 .



theory monoid
U, *, e,
axiom_closure
axiom_assoc
unital_set?j
theory unital_set
j : monoid
U' := subset of U with inverses
group?i
theory group
i : monoid
axiom_inv
view m : group ⟼ unital_set
i / U ⟼ U'
i / * ⟼ j / *
i / e ⟼ j / e
i / axiom_closure ⟼ (proof that it holds in unital_set)
axiom_inv ⟼ (proof that it holds in unital_set)

Figure 1: Theory graph γ comprising six theory modules as nodes, with imports between them as edges

Symbol level: The imports (**unital_set?j** and **group?i**) are marked by solid arrows. Each import carries all the symbols from the source theory to the destination. In theory **monoid**, the following constants are declared:

- U - underlying set
- * - monoid operation
- e - unity element
- axiom_closure, axiom_assoc

Theory **group** imports from monoid via the structure **i** and formulates the axiom of invertibility. Theory **unital_set** imports from monoid via **j** and declares **U'** - the set of invertible elements in U.

Note that Figure 1 is very informal, since it only shows the names of the constants, omitting their types and definiens, and does not define the structure bodies.

In general, a constant has the form $c : \tau := \delta$, where c is the unqualified name of the constant, τ is its type and δ its definiens. Both τ and δ are optional; $\perp$ is used to signal omission.

A structure from S to T, such as **group?i**, maps *all* objects of S to terms over T. As opposed to views, which relate fixed theories without changing either one, structures change the target theory T by importing all symbols from S to T. Thus, they have definitional flavor

Object level: According to the Curry-Howard correspondence, each axiom F is introduced by a constant of type $trueF$ and no definiens, where $true$ is defined in the meta-theory.

An extension to MMT was developed in [Aga08], which suggests a way to represent functors.

A functor, as a concept in category theory, associates to each object from a category C an object from category C' and associates to each morphism from category C a morphism from category C', while preserving the identity and the composition of morphisms. In MMT , theories play the role of categories and morphisms between theories play the role of objects. Functors are then represented as λ-abstractions that can take morphisms as parameters, as shown in Figure 2 .

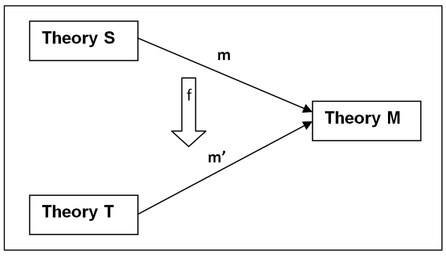


Figure 2: General form of a functor in [Aga08]

The syntax suggested was:

$$
\begin{array}{lll}
m' & := & \lambda m : S \to M \ \ . \ \ (\\
 & & \quad \ldots \\
 & & \quad c_i \to \omega_i \\
 & & \quad \ldots \\
 & & \quad) \ \ : T \to M
\end{array}
$$

The functor m' can then be applied to a morphism $\mu : S \to M$ matching the "variable" m. Thus, $m'(\mu)$ becomes a valid morphism from T to M.

3 Motivation

'

Our work was inspired by several practical necessities in representing Mathematics, observed in Algebra I [Bou89]. The issues discussed below have been implemented in Section 4 .

3.1 Morphism quantification

Although Figure 1 is inspired from Algebra I [Bou89], the actual definition of *group* is a bit different in the book [Bou89, Def. 6, Page 15]:

> Let E be a unital magma. [...] An element x of E is called *invertible* if it has an inverse.
> A monoid all of whose elements are invertible is called a group.

A more appropriate representation of the text above is via the theory $monoid\_with\_invertible\_element$: we would like to be able to define $?group?axiom\_inv$, informally, like this:

$$axiom\_inv : true \quad \forall x : i,\ \exists(\chi : monoid\_with\_inv\_element \mapsto group := \{u \mapsto x\})$$

Moreover, we would like this to have, informally, the same meaning as the following declaration (where I is a shorthand for $monoid\_with\_inv\_element$):

$$\begin{aligned} axiom\_inv : true \quad & \forall x : i,\ \exists i' : sort,\ \exists *' : i' * i' \mapsto i', \exists e' : i',\ \exists u' : i', \\ & i' = mon/magma/i \text{ and } *' = mon/magma/* \text{ and } e' = mon/magma/e \text{ and } u' = x \\ & \text{ and } \forall g : i', g *' e' = g \text{ and } \exists u^{-1},\ u' *' u^{-1} = e'. \end{aligned}$$

In other words, we encountered the need to quantify over morphisms. As it turns out, in order to apply such a transformation, one needs to know beforehand exactly which constants and imports are explicitly mapped by chi. Those who are not will automatically become variables, and are assigned a type. However, no further conclusion is inferred about these variables.

3.2 Better functors

Although [Aga08], discussed in Section 2 , offers a simple syntax and semantics for functors, the paper has several limitations:

- First, the idea is implemented to a reduced MMT language with no imports; we would like to extend the notion to the full language.
- Secondly, the λ-abstractions can only have one morphism argument; however, mathematicians often employ functors which take multiple morphisms as argument (i.e. product of 2 or more categories).

In Section 4 , we will alter the syntax of views to accommodate multiple variable morphisms, making them more useful.

3.3 Applying terms to morphisms

One also encounters expressions involving morphisms, mainly logical judgements. Since this is quite common in Mathematics, MMT should be powerful enough to express them. A typical situation is to have construct a concept in two different ways, and then to prove their equivalence. The $group$ defined in the previous example can obviously be defined with the usual group axioms, without involving any other algebraic concept. Let's call the latter axiomatic definition $group'$:

$theory\ group' = \{$
universe set,
$+$ and $\cdot$ operations
neutral element
axioms for closure, associativity, neutral and inverse element
$\}$

The natural way to express the equivalence of $group$ and $group'$ in MMT is to declare two views,

$$\begin{aligned} \mu_1 &: group \rightarrow group' \\ \mu_2 &: group' \rightarrow group, \end{aligned}$$

and then to write a theorem

$$\mu_1 \circ \mu_2 = id_{group}$$

As we will see, this is possible and desirable to introduce, albeit with a caveat: the entire expression needs to be translated in the current theory T, via a morphism $\mu : group \to T$. This is then a valid theorem in T stating that $group$ and $group'$ are equivalent notions:

$$c : (\mu_1 \circ \mu_2 = id_{group})^{\mu} := proof,$$

where μ is an existing morphism from $group$ to T. According to the Curry-Howard correspondence in use, c is the name of the theorem, the type of c is the actual theorem statement and the definiens is the proof, represented appropriately using terms.

4 Reasoning over Morphisms

4.1 An Extended Grammar for MMT

We attempt to formalize the new MMT syntax with morphism variables. Table 1 defines the augmented grammar in BackusNaur Form with the additional operators *, $^+$ and $[\_]$, which denote repetition, non-empty repetition and optional part, respectively. The nonterminals have been colored in blue for better visibility.

The syntax defined in this grammar does not ensure the uniqueness of names, correct typing of assignments, or well-formedness in general. Such restrictions are imposed using the sequent calculus rules, which will be sketched in Appendix A . Having these properties imposed by the grammar would render the grammar much too complex and inefficient.

Next section explains, one by one, the new concepts introduced in Table 1 .

4.2 Intuitions Behind the New Syntax

We summarize the main points in our implementation of the issues outlined in Section 3 . The exact semantics are given by the judgements, several of which are sketched in Appendix A .

4.2.1 Functor-like views

In order to represent functors, we allowed each view to have a list of structures at the beginning, ϑ:

$$Viw ::= m : T \to T := \{\vartheta; \sigma\} \ \mid \ m : T \to T := \mu$$

Although the syntax is the same, the semantics of these structures is different from those embedded in theories. Here, structures with empty body, $i : S := \{\cdot\}$, play the role of **parameters** to the view. Then, $m\{\dots, i \to \mu, \dots\}$ is a valid morphism, which is normalized by replacing each occurrence of i with μ.

To increase the similarity between ϑ in view and ϑ as theory body, we allow structures with non-empty body $i : S' := \{\sigma\}$, and even of the type $i : S' := \mu'$. The definitions inside σ (or those induced by μ) are then treated as **obligations** that the substituting morphism μ has to obey. Namely, for each constant $\vec{c}$ whose definiens $\omega \neq \bot$ is provided by i, we need to have:

$$\omega \equiv \vec{c}^{\,\mu}$$

Explanation	Nonterminal		Definition
Theory graph	γ	::=	$(Thy \mid Viw)^*$
Theory	Thy	::=	$T := \{\vartheta\}$
View	Viw	::=	$m : T \to T := \{\vartheta; \sigma\} \mid m : T \to T := \mu$
Theory body	ϑ	::=	$(Con \mid Str)^*$
Link body	σ	::=	$(ConAss \mid StrAss)^*$
Constant	Con	::=	$c\ [: \omega]\ [:= \omega]$
Structure	Str	::=	$s : T := \{\sigma\} \mid s : T := \mu$
Assignment to constant	$ConAss$	::=	$\vec{c} \mapsto \omega$
Assignment to structure	$StrAss$	::=	$\vec{s} \mapsto \mu$
Morphism	μ	::=	$id_T \mid \vec{s} \mid X \mid \mu \circ \mu \mid m\ \{\sigma\}$
Term	ω	::=	$\top \mid T?\vec{c} \mid x \mid \omega^{\mu} \mid @(\omega, \omega^+) \mid \beta(\omega; \Upsilon; \omega) \mid @(\omega; \omega; \mu^+)$
Context	Υ	::=	$X : T \to T\ := \{\sigma\} \mid X : T \to T := \mu \mid \vec{x}\ [: \omega]\ [:= \omega]$
Module reference	T	::=	URI?T
	m	::=	URI?m
Qualified identifier	$\vec{c}$	::=	$c \mid s/\vec{c}$
	$\vec{s}$	::=	$s \mid s/\vec{s}$
	$\vec{x}$	::=	$x \mid X/\vec{c}$
Unqualified identifier	T, m, c, s, x, X	::=	pchar $^+$
	URI	::=	URI, no query, no fragment (see RFC 3986 [T. 05])

Table 1: The new grammar for MMT expressions.

Suppose that $v : S \to T$ is a view with a parameter $i : S' := \_$ inside and that we want to have a morphism $m\{i \to \mu\}$. Then μ must be a morphism from S' to T and the above rules must hold, for every such $\vec{c}$:

$$\frac{\gamma >\!\!>_m \vec{c} : \tau := \omega}{\gamma \rhd_T \omega \equiv \vec{c}^{\,\mu}}$$

and

$$\frac{\gamma >\!\!>_m \vec{c} : \tau := \omega}{\gamma \rhd_T \vec{c}^{\,\mu} : \tau}$$

The second rule says that $\vec{c}^{\,\mu}$ must be of type τ. For most practical purposes, this will automatically hold whenever the first equality condition holds. We keep it though for maximum generality.

For greater flexibility and to induce a certain symmetry between morphism and term variables, we also introduce the usual constant declarations in ϑ. Declarations of the type $c : \tau := \omega$, with $\omega \neq \bot$, can be useful shortcuts in case the view contains extensive proofs. Declarations of the type $c : \tau := \bot$ and scalar parameters, which have to be instantiated just like morphism parameters when constructing a morphism. Thus, if we have

$$m : S \to T := \{\ldots, c : \tau, \ldots; \sigma\},$$

then we can form a morphism via

$$m\{\ldots, c \to \omega, \ldots\}.$$

The type τ always constitutes an obligation (intuitively, the argument must match the parameter type):

$$\gamma \rhd_T \omega : \tau.$$

In the case $c : \tau := \omega$ (c has a definiens in m), it is not mandatory to provide a substitution $c \to \omega'$ in $m\{\sigma'\}$. Nevertheless, if the substitution is present, then ω constitutes an obligation and we need to have

$$\gamma \rhd_T \omega \equiv \omega'$$

and

$$\gamma \rhd_T \omega' : \tau$$

for each such c.

4.2.2 Binding morphisms

In Section 3 , we discussed the importance of morphism bindings and we presented an example. We decide to introduce *morphism variables* with syntax similar to both types of links - views and structures: $X : S \to T := \{\sigma\}$ and $X : S \to T := \mu$. The next step is to allow them to appear in contexts, alongside variables, and to introduce quantification via the existing β rule. Surprizingly, the syntax of β-expressions doesn't have to change:

$$\beta(\omega; \Upsilon; \omega)$$

However, the semantics has to be established. The best way to do this is to show how morphism variables can be eliminated from a document fragment, by replacing them with existing objects (this process is otherwise known as normalization and will be exposed in detail in Section 4.3). The process is as follows:

1. We require T to be the home theory of the term in which the β-expression occurs. This gives the opportunity to make X as similar as possible to a structure, reducing in this way the language complexity.
2. For each constant $\vec{c}$ in S for which X doesn't provide an assignment, we create a new variable $X/\vec{c}$, which will be the result of the normalization of $S/\vec{c}^{\,X}$. This is completely similar to the case of undefined constants in structures, denoted $i/\vec{c}$. All the new variables create in this way are then introduced in the context, instead of the morphism variable.
3. For each constant $\vec{c}$ in S for which X provides an assignment ω, we don't create a new variable, but simply allow the notation $X/\vec{c}$, which normalizes to ω.

4.2.3 Typed variables

In the original OMDoc design, variables were only described by their name, thus breaking the symmetry between terms (who have types and definiens) and variables. This design was also inherited by MMT .

However, Mathematical texts often employ typed variables:

For every set x . . .
Every group G has the property...

and variables with both definition and type:

Let x be the minimum solution...

$$\sum_{k=0}^{\inf} \dots$$

$$\lim_{x \to 0} \dots$$

The only way to represent these formulas was via attributions $\alpha(\omega; \omega = \omega)$. This was somewhat counter-intuitive and allowed too much generality. Furthermore, the MMT system was no longer able to verify correct typing for expressions involving variables, relying too much on the underlying foundation in this respect.

By introducing optional types and definitions for variables ($x[: \tau] := [\omega]$), the symmetry would be re-established.

As an added bonus, the contexts have now a unified syntax for normal and morphism variables, the letter having a type ($S \to T$) and a ¨definiens¨(either $\{\sigma\}$ or μ).

Last, but not least, we decide to remove the α-attributions, since they do not add to the language expressivity any more; their primary role was binding variables.

4.2.4 Applying terms to morphisms

Section 3.3 has outlined the importance of dealing with morphisms themselves instead of just morphism applications to terms.

Therefore, we decide to represent formulas like $\mu = \mu'$, $\mu \neq \mu'$, $\mu \leq \mu'$, ... via a special application

$$@(f; g; \mu_1, \dots, \mu_n),$$

where $\mu_1, \dots, \mu_n$ have the same domain theory S with induced constants c_1, ..., c_k and the same codomain T, which must coincide with the home theory of the formula. f and g are user-defined terms.

The application has the following normalization:

$$f\left(g\left(c_1{}^{\mu_1}, \dots, c_1{}^{\mu_n}\right), \dots, g\left(c_k{}^{\mu_1}, \dots, c_k{}^{\mu_k}\right)\right)$$

or, more formally,

$$@\left(f; @\left(g; c_1{}^{\mu_1}, \dots, c_1{}^{\mu_n}\right), \dots, @\left(g; c_k{}^{\mu_1}, \dots, c_k{}^{\mu_n}\right)\right).$$

For example, the equality $\mu_1 \circ \mu_2 = id_{group}$ from Section 3.3 can be written as

$$@\left(and; =; \mu_1 \circ \mu_2, id_{group}\right),$$

where

- "*and*" is a user-defined term taking a variable number of arguments (must be able to take as many arguments as there are induced constants in *group*)
- "=" is a user-defined term taking two arguments (during the normalization, it will be used as $@(=; c_j{}^{\mu_1 \circ \mu_2}, c_j{}^{id_{group}})$, for each j).

4.3 Normalization

Let γ be a well-formed theory graph and ω a well-formed term over γ, home theory T and context Υ. We propose a way to remove from ω all structures, morphism variables and morphism applications, by writing the formula in terms of the "classic" MMT symbols.

The normalization of the term ω is denoted by $\overline{\omega}\,^{\gamma}_{\Upsilon}$, or simply $\overline{\omega}_{\Upsilon}$, where no confusion may arise, and is defined by induction over the structure of ω:

$$\begin{aligned}
\overline{\top}_{\Upsilon} &:= \top \\
\overline{T?\vec{c}}_{\Upsilon} &:= \begin{cases} \overline{\delta}_{\Upsilon} & \text{if } \gamma \gg_T \vec{c} : \_ := \delta,\ \delta \neq \bot \\ T?\vec{c} & \text{otherwise} \end{cases} \\
\overline{\vec{x}}_{\Upsilon} &:= \begin{cases} \overline{\delta}_{\Upsilon}, & \text{if } \vec{x} : \_ := \delta \neq \bot \in \Upsilon \\ \vec{x}, & \text{otherwise} \end{cases} \\
\overline{@(\omega_1, \ldots, \omega_n)}_{\Upsilon} &:= \begin{cases} @\left(\overline{\omega_1}_{\Upsilon}, \ldots, \overline{\omega_n}_{\Upsilon}\right) & \text{if } \overline{\omega_i}_{\Upsilon} \neq \top \text{ for all } i \\ \top & \text{otherwise} \end{cases} \\
\overline{\vec{x} : \tau := \delta,\ \Upsilon'}_{\Upsilon} &:= \vec{x} : \overline{\tau}_{\Upsilon} := \overline{\delta}_{\Upsilon}, \overline{\Upsilon'}_{\Upsilon, \vec{x} : \overline{\tau}_{\Upsilon} := \overline{\delta}_{\Upsilon}}.
\end{aligned}$$

A more interesting case is the translation of constants along a morphism. Suppose $m : S \to T$, $\gamma \gg_S \vec{c} : \tau := \delta$, $\gamma \gg_m \vec{c} \mapsto \delta'$, $v : S \to T := \{\gamma, \sigma\}$, $\gamma \gg_{v\{\sigma'\}} \vec{c} \mapsto \delta'$.

$$\overline{(S?\vec{c})^m}_{\Upsilon} := \begin{cases} \overline{\delta^m}_{\Upsilon}, & \text{if } \delta \neq \bot, \delta' = \bot \\ \overline{\delta'}_{\Upsilon}, & \text{if } \delta' \neq \bot \\ \top, & \text{if } \delta = \delta' = \bot \text{ and } m = v \text{ view} \\ \overline{T?\vec{s}/\vec{c}}_{\Upsilon}, & \text{if } \delta = \delta' = \bot \text{ and } m = T?\vec{s} \text{ or } m = v\{\sigma'\}?\vec{s} \text{ structure} \end{cases}$$

The special cases of translation along a variable, β-reductions and @-applications for morphisms are ommitted; however, the last two are informally discussed in Section 4.2 .

Once normalization is done on all terms, **flattening** can be performed, i.e. the elimination of structures from theories and declarations from views. Flattening has two benefits:

1. It proves that our additions to MMT , while providing more power of expression, can still be translated into plain OMDOC and thus, all the existing utilities for OMDOC and MMT can still be used successfully.
2. It is the obvious way to check that two objects (terms, morphisms, theories, theory graphs) are semantically equal, by reducing them to the same set of constants and variables, unique up to renaming.

Normalization and flattening have a huge caveat: the exponential blow-up in size, compared to the original MMT fragment. The rules can be rewritten, however, to take advantage of the modular structure of MMT : instead of using fully normalized terms, they may only do normalization up to a level and then check whether a direct rule can satisfy the goal [Rab08].

4.4 Conclusions and Outlook

Our work introduced several extensions to the MMT module system for mathematical theories, a simple, but expressive language for logical knowledge exchange. MMT documents are collections of theories (collections of constant declarations and imports from other theories) and views, which assign symbols from one theory with terms constructed over another.

View traditionally play the role of theorems. We propose a different kind of views - inspired by functors - which can take other morphisms as parameters, by having a list of (not fully defined) imports before the assignments. In addition to imports, we allow constant declarations inside views, in order to establish a symmetry between the two kinds of modules.

We further expand the language by allowing morphism variables inside contexts and even inside terms. Quantification and reasoning over morphisms is kept independent of foundation and is fully customizable by the user. Thus, MMT remains suitable as an universal logical exchange format, even though the language capabilities are expanded.

Finally, the $alpha$-attributions for terms are dropped in favor of typing and definitions for variables. This unclutters the syntax, while allowing better well-formedness checking.

We also show that morphism variables and applications can be eliminated via normalization. This suggests that flattening is still possible with the new language.

References

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Minerv 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, 2003. See http://www.w3.org/TR/MathML2/.

[Aga08] E. Agapie. Functors in a web-scalable module system. *Guided research report*, Jacobs University, Bremen, 2008.

[BCC+04] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open-Math Standard, Version 2.0. Technical report, 2004. See http://www.openmath.org/standard/om20.

[Bou89] N. Bourbaki. *Algebra I*. Elements of Mathematics. Springer, Softcover edition of the 2nd printing edition, 1989. Translation from French.

[CF58] H. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.

[Gor88] M.J.C. Gordon. Hol: A proof generating system for higher-order logic. In *VLSI Specification, Verification, and Synthesis*, pages 73–128. Kluwer Academic, dordrecht edition, 1988.

[How80] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[Koh08] M. Kohlhase. An Open Markup Format for Mathematical Documents, OMDoc [Version 1.2]. Technical report, 2008. See http://www.omdoc.org/pubs/omdoc1.2.pdf.

[Odl94] A. M. Odlyzko. Tragic loss or good riddance? the impending demise of traditional scholarly journals. *Journal of Universal Computer Science*, pages 3–52, 1994. See http://www.jucs.org/jucs_0_0/tragic_loss_or_good.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover (Lecture Notes in Computer Science)*. Springer, 1994.

[Rab08] F. Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University, Bremen, 2008. See http://kwarc.info/frabe/Research/phdthesis.pdf.

[Rab09] F. Rabe. The MMT language. Jacobs University, 2009. See https://kwarc.eecs.iu-bremen.de/repos/kwarc/rabe/Papers/omdoc-spec/MKM_09/paper.pdf.

[RK08] F. Rabe and M. Kohlhase. An Exchange Format for Modular Knowledge. 2008. number 418.

[T. 05] T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force, 2005. RFC 3986. See http://www.ietf.org/rfc/rfc3986.txt.

A Well-formed Expressions

$$\frac{\cdot}{\rhd\ \cdot}\ \mathit{Start} \qquad\qquad \frac{\rhd\ \gamma}{\rhd\gamma,\ T := \{\cdot\}}\ \mathit{Thy}$$

$$\frac{\rhd\gamma \qquad \rhd\ \mu : S \mapsto T}{\rhd\gamma,\ \ v : S \mapsto T\ \ := \{\cdot\}}\ \mathit{ViewDef}$$

$$\frac{\rhd\gamma \qquad T, S \in \mathit{Thy}(\gamma)}{\rhd\gamma,\ \ v : S \mapsto T\ \ := \{\cdot\}}\ \mathit{View}$$

Figure 3: Adding Modules

$$\frac{\rhd\gamma,\ \ T := \{\sigma\} \qquad \gamma \rhd_T\ \ \delta : \tau}{\rhd\gamma,\ T := \{\sigma, c : \tau := \delta\}}\ \mathit{Con}$$

$$\frac{\rhd\gamma,\ \ T := \{\sigma\} \qquad \gamma \rhd\ \ \mu : S \mapsto T}{\rhd\gamma,\ T := \{\sigma, s : S := \mu\}}\ \mathit{StrDef}$$

$$\frac{\rhd\gamma,\ \ T := \{\sigma\} \qquad S \in \mathit{Thy}(\gamma) \setminus \{T\}}{\rhd\gamma,\ T := \{\sigma, s : S := \{\cdot\}\}}\ \mathit{Str}$$

$$\frac{\rhd\gamma,\ \ v : S \mapsto T := \{\vartheta\} \qquad \gamma \rhd_T\ \ \delta : \tau}{\rhd\gamma,\ v : S \mapsto T := \{\vartheta,\ c : \tau := \delta\}}\ \mathit{LinkCon}$$

$$\frac{\rhd\gamma,\ \ v : S \mapsto T := \{\vartheta\} \qquad S' \in \mathit{Thy}(\gamma)}{\rhd\gamma,\ v : S \mapsto T := \{\vartheta,\ s : S' := \{\cdot\}\}}\ \mathit{LinkStr}$$

Figure 4: Adding Symbols

$$\frac{\gamma;\Upsilon >> \; m : S \mapsto T :=}{\gamma;\Upsilon \rhd \; m : S \mapsto T} \; \mathcal{M}_{induced}$$

$$\frac{T \in Thy(\gamma)}{\gamma;\Upsilon \rhd \; id_T : T \mapsto T} \; \mathcal{M}_{id} \qquad \frac{\gamma;\Upsilon \rhd \; \mu : R \mapsto S \qquad \gamma;\Upsilon' \rhd \mu' : S \mapsto T}{\gamma;\Upsilon,\Upsilon' \rhd \mu_0\mu' : R \mapsto T} \; \mathcal{M}_{,}$$

$$\frac{\begin{array}{c} v : S \mapsto T := \vartheta, \sigma \in \gamma \qquad \gamma;\Upsilon \rhd_T \delta' : \tau \\ \gamma;\Upsilon \rhd_T \; \delta \leq \delta' \qquad \gamma^v_{sym}(c) = (\tau,\delta) \qquad \gamma;\Upsilon \rhd^{part} v\sigma' : S \mapsto T \end{array}}{\gamma;\Upsilon \rhd^{part} \; v\{\sigma', c \mapsto \delta'\} : S \mapsto T} \; \mathcal{M}^{part}_{con}$$

$$\frac{\begin{array}{ccc} & & [[\gamma >>_{S'} \vec{c} : \tau := \delta]] \\ v : S \mapsto T := \{\vartheta,\sigma\} \in \gamma & \gamma;\Upsilon \rhd^{part} v\{\sigma'\} : S \mapsto T & \vdots \\ \gamma;\Upsilon' \rhd \mu : S' \mapsto T & s : S' := \{\sigma''\} \in^{\gamma}_{sym} v & \\ & & \gamma \rhd_T \vec{c} \end{array}}{\gamma \rhd^{part} \; v\{\sigma', s \mapsto \mu\} : S \mapsto T} \; \mathcal{M}^{part}_{str}$$

Figure 5: Morphisms