Injecting Formal Mathematics Into LaTeX

Dennis Müller[0000-0002-4482-4912] and Michael Kohlhase[0000-0002-9859-6337]
{dennis.mueller,michael.kohlhase}@fau.de

Computer Science, FAU Erlangen-Nürnberg
https://kwarc.info

Updated 2023-04-25: Syntax for sTeX 3.3

Abstract. The paper presents the sTeX3 format for representing informal mathematics. sTeX3 acts as a surface language for two systems: the (presentation-oriented) LaTeX system to produce PDF and the semantics-aware Mmt system for advanced knowledge management services. We discuss how the sTeX3 markup facilities allow in situ flexiformalization (and the necessary elaboration of complex structures), while staying presentationally neutral.

This paper uses sTeX3. The semantically annotated XHTML version of this paper is available at `https://url.mathhub.info/cicm22stexmmt`

1 Introduction

Most formal systems offer two kinds of representation formats for input/output: a standardized system language for efficient storage and loading of compiled content and various surface languages for interacting with users. While the former usually optimize fully explicit representations, the latter cater to the notational preferences (and cognitive abilities) of humans. As an effect, surface languages usually have to be elaborated into the system language by e.g. macro expansion, overloading resolution, and type inference. This setup works very well where there is only one formal system that consumes the surface language.

For sTeX [Koh08; sTeX] however, the LaTeX-based surface language for Mmt/OMDoc presented in this paper, we have two such consumers: The (presentation-oriented) LaTeX system to produce PDF, and the semantics-aware Mmt system [RK13; MMT] for advanced knowledge management services.

With our recent redesign and reimplementation of sTeX as sTeX3 (see our System Description at this very conference), we decided on two criteria regarding these two consumers:

LaTeX: sTeX3 should be compatible with arbitrary packages, document classes and general styles of presentation and typesetting, without impacting an authors preferred mode of expression or the final document layout, while still allowing for annotating text fragments with their (flexi-)formal semantics, and

Mmt: sTeX3 should act as a functionally complete surface language for the Mmt system.
This implies two important challenges to solve:

Informal→Formal: sTeX3 needs to provide support for annotating mathematical content *as actually written* by authors in practice – i.e. it needs to support the linguistic phenomena ubiquitously employed by (informal) mathematicians, and appropriately translate them into (flexi-)formal presentations actionable by Mmt.

Formal→Informal: sTeX3 needs to allow for all the structural and object-level features provided by the Mmt system, without being (too) invasive with respect to the final presentation of the content, and (importantly) without being semantics-aware and having access to the associated algorithms of a formal system (e.g. type inference), to remain consumable by LaTeX.

In the following, we describe our approach to solving these two challenges in Sections 3 and 4, respectively.

This paper itself uses sTeX3 within the `llncs` document class. We therefore made the source files available on Overleaf at `https://www.overleaf.com/read/bztqrgbjmkcd`. The semantically annotated XHTML generated from it is linked above; the Mmt/OMDoc declarations extracted and checked by Mmt can be browsed at `https://tinyurl.com/cicm22stexomdoc`[1].

2 Preliminaries

Both sTeX and Mmt are based on the OMDoc [Koh06] format (Open Mathematical Documents), a conceptual ontology and XML-based representation format for the content and document structure of (informal) mathematical documents. sTeX is the result of adding semantic macros to LaTeX to annotate the OMDoc markup without changing the generated PDF, while Mmt re-engineers and implements the formal core of OMDoc, using Mmt/OMDoc as an ontology and XML-based system language and implements it in the Mmt system (and corresponding surface language).

We will now give a brief overview over the Mmt/OMDoc ontology and discuss how sTeX3 now implements it in the form of a LaTeX package. For system-related details, see the companion system description published at CICM 2022.

Mmt/OMDoc*:* For the detailled ontology of Mmt/OMDoc, we refer to [Mül19]. For now we restrict ourselves to the basics: content in Mmt/OMDoc is organized in named theories, containing declarations (primarily constants). A constant has a name, optional aliases, an optional type and an optional definiens. Type and definiens are represented as OpenMath expressions. Theories can

[1] Note that the latter requires a browser with MathML support to display formal expressions, such as Firefox, but notably not (vanilla) Chrome *yet* (intent-to-publish for MathML in Chromium has been announced, expected in 2023).

include other theories, making their declarations available in the including theory. The semantics of MMT content is provided by rules written in Scala and activated in theories via dedicated rule constants, providing their fully qualified class path and (optionally) parameters. All theories and declarations are assigned globally unique MMT URIs for unambiguous referencing.

We will introduce and discuss the remaining concepts of the MMT/OMDoc ontology in their respective dedicated sections below.

STEX: Omitting the details on how structured data can be represented in TEX, **begin**{smodule}{name} opens a new module (the STEX equivalent of a MMT theory). It is represented internally as several macros storing data such as its name, its included modules, the names of contained symbols, and code to execute whenever the module is *activated* (e.g. when being included in some other module).

\symdecl{macroname}[name=symbolname,type=tp,def=df,args=arity] generates a new symbol with provided arity, name, type and definiens and generates a semantic macro *macroname* that expands to \invoke_symbol{namespace?module?symname}. The command \notation{symname}[id]{...} introduces a new notation for a symbol with identifier `id`, that \invoke_symbol can defer to for typesetting, e.g. when using *macroname*[id]. Importantly, "generating a new symbol" is distinct from generating a semantic macro, a symbol's name is distinct from its semantic macro, and notations are internally separate from the symbol they are attached to, allowing for them to be copied and inherited between symbols. The macro \symdef combines \symdecl and \notation for convenience.

STEX content can be formatted by `pdflatex`, and additionally be converted to semantically annotated XHTML, the annotations of which are entirely dictated by the STEX package itself. Most STEX macros and environments, when being converted to XHTML, introduce annotations in the form of XML attributes indicating their corresponding MMT/OMDoc concept – for example, a notation introduces annotations for the OpenMath label (e.g. OMA, OMID, OMBIND), the full MMT URI of the symbol the notation belongs to, the argument positions, etc. As a result, the MMT system can import the generated XHTML by (to a large extent) directly mapping OpenMath-related annotations to actual OpenMath expressions, annotations generated by \symdecl to MMT constants, and those generated by **begin**/**end**{smodule} to MMT theories.

The macro \importmodule[some/archive]{path?Modulename} includes the module `Modulename` from the (optionally) provided archive into the current module, and exports an appropriate annotation with the full MMT URI in the XHTML, which MMT can directly translate to an include of the corresponding MMT theory.

The macro \MMTrule{classpath}{arguments} is largely ignored in STEX except for introducing a corresponding annotation that MMT can translate to a rule constant.

As in MMT, each module can be assigned a *meta-theory*; if none is specified, a default meta-theory is used. For the purposes of this paper, only two of its symbols are relevant: *bind* $((\cdot) \to \cdot)$ behaves like (an informal variant of) a lambda or (dependent function type) pi operator, depending on context; Up-

dated 2023-04-25: syntax\symuse{implicit bind} (implicitbind) behaves largely the same way, but marks the variable(s) bound as representing *implicit* arguments.

The rest of this paper will now focus on the remaining features of sTeX not directly covered by Mmt/OMDoc, and the remaining features of Mmt not covered by sTeX by all of the above.

As a continuous example thoughout, we will work our way towards defining a *lattice* as a mathematical structure composing two *semilattices*, and declare the associated duality principle and the equivalence of the algebraic and order-theoretic definitions as theory morphisms, such that the presentation remains "typical" for informal mathematics and the extracted formal content can be fully type checked by Mmt, all in-situ in (the source code of) this very paper.

3 Translating Phenomena in Informal Mathematics to a Formal System

3.1 Formal Theories vs. Document Fragments

The most fundamental difference between content representation formats for informal mathematics like OMDoc and for formal mathematical knowledge like Mmt is that the former focuses on structured (fragments of) mathematical text whereas the latter focus on modular collections of assertions of properties of objects (called declarations in Mmt). Both OMDoc and Mmt use OpenMath expressions to represent the (functional structure) of objects, i.e. mathematical formulae, and theories/inheritance to represent contexts.

While mathematical knowledge in Mmt can be structured and sequentialized according to visibility criteria alone (i.e. which declarations can be referenced in some context), sTeX needs to take the narrative structure of documents into account. In particular, there must be a facility to make module contexts available in document fragments to make the referenced concepts in them well-scoped. But unlike theories, they should not always "re-export" the used declarations, since this can lead to circular dependencies. For instance, in motivational or "preview" fragments, mathematical concepts are "used" before they are introduced. To account for this, sTeX provides the \usemodule instruction which behaves like \importmodule but without "exporting" the contents of the used module when the current module is imported elsewhere. This distinction however does not exist in Mmt.

Another important distinction of formal and informal systems is that informal mathematics relies on mathematical vernacular that comes in natural language variants, whereas formal systems are – a priori – (natural) language agnostic.

Signature and Language Theories To account for this language diversity we generate two distinct Mmt theories from an sTeX-module: The *signature theory* is assigned the actual Mmt URI `ns?name` of the module it is generated from,

and contains exactly the formal, declarative parts of the module: The includes and constants generated from \importmodules and \symdecls, respectively.

An additional *language theory* is assigned the MMT URI `ns/name?lang`, where `lang` is the language the original document is in (by default english). This theory includes the signature theory, an explicit include for every usemodule, and for every (non-trivial) OPENMATH-expression occurring outside of symbol components a constant with that expression as definiens (which is checked by MMT by having its type inferred).

Besides allowing for representing all aspects of a document with a formal semantics, this distinction between language and signature theories is also very well suited for STEX3's approach to multilinguality in general (see [KM] for details).

3.2 Implicit Arguments and Variables

Another – often overlooked – difference between formal and informal systems is the use of variables. In MMT, declarations are self-contained (i.e. closed) formal expressions, whereas OMDOC statements are text fragments containing (usually/often open) expressions. For instance, an english sentence like "For a natural number n, $n+n$ is even." contains the two open expressions n and $n+n$, whereas the corresponding formal declaration $\vdash \forall n : \mathbb{N}.\text{even}(n+n)$ is closed.

Thus in MMT, variables only ever occur on the object-level, i.e. in a type or definiens component of a term, and bound by some binding operator.

In OMDOC, the majority of symbols are arguably variables, by virtue of being unspecified, postulated or can be thought of as "universally quantified" over whole paragraphs, chapters or even full documents as scope. In the example above, the fragment "natural number n" acts as a variable declaration, whereas the "n" in the second expression only references the variable n.

Correspondingly, STEX3 allows for using variables in two ways: The command \svar[name=x]{some variable} marks up the text some variable as being a reference to some variable with name `x`. This is mostly useful for simple variable names, e.g. \svar{x}. Additionally, we can *declare* a variable including its notation, type, definiens etc. in the same manner as we do symbols, using \vardef{setA}[type=\collection]{\comp{A}}, which allows us to invoke the variable using the semantic macro \setA, producing A. This macro is local to the current TEX-group, determining its scope in the document. Such a variable declaration is exported to the XHTML and used by MMT when resolving its subsequent occurrences.

For any expression we want to convert to MMT/OMDOC, we look up any free variables in the current scope to determine their types and other attributes (if given), and abstract them away using a dedicated binding operator Updated 2023-04-25: syntax\symuse{implicit bind}, effectively marking them as *implicit arguments*, which is handled thusly: Whenever a term with head o occurs in an XHTML document, MMT looks up its type and/or definiens. If either is of the form \symuse{implicit bind}{vars}{bd}, MMT inserts fresh variables in their place and attempts to infer their values during type checking. This makes sure that

all variables in an STEX document are bound and also allows us to conveniently specify implicit arguments. For example, we can declare typed equality like this:

```
\vardef{varx}[type=\setA]{\comp x}
\vardef{vary}[type=\setA]{\comp y}
\symdef{tpeq}[name=typed equality,args=2,
  type=\bind{\varx,\vary}\prop]{#1 \comp= #2}
```

The explicit type we wrote down here is simply (x, y)→Prop. However, by having declared x and y to be of type A, MMT considers A to be a free variable and abstracts it away, yielding the actual type $\{(A \in \mathtt{SET})\}_I(x \in A, y \in A)$→Prop.

This allows for writing the types of symbols in accordance with their (notational) arity, practically hiding implicit arguments from an author entirely.

While it might seem cumbersome to explicitly declare variables rather than just typing \svar{x} or even x directly (which both STEX and the subsequent MMT-import allow for as well), in practice it has turned out that variables will be reused often enough (see e.g. the source file of this paper) to not cause too much overhead, while allowing for declaring their types and other attributes *once*.

3.3 Flexary Operators and Argument Sequences

While in formal settings, symbols are usually *required* to have a fixed arity, *flexary* operators (or, rather, notations) abound in informal settings. In most cases, this can be seen as a mere notational convention, e.g. if addition is known to be associative, then writing $a + b + c$ is unambiguous; similarly, by convention $A \to B \to C$ represents $A \to (B \to C)$ rather than $(A \to B) \to C$. Nevertheless, restricting ourselves to binary operators would mean forcing authors to type e.g. \plus{a}{\plus{b}{c}}, which would be prohibitively cumbersome. Even worse: If we wanted to write $a = b = c$, it is not even clear how to do that given a binary equality alone, given that e.g. the subexpression $a = b$ is a proposition, and the statement clearly does not mean to assert that *the proposition* $a = b$ is equal to c.

[HKR14] attempts to solve this problem specifically in the context of MMT, however, here the notion that an operator is *flexary* (taking a sequence as an argument) is reflected both in the type and the definiens of a symbol – contrary to the idea that e.g. equality should "formally" be a *binary* operation with corresponding type, regardless of (flexary) "abuse" of notation.

Syntactically, STEX handles this problem by allowing symbols to take a comma-separated list as (one or several) arguments, signified via an a in its key args= That allows us to specify equality more adequately as: Updated 2023-04-25: Syntax

```
\symdef{eq}[name=equal,args=a,
  type=\bind{\varx,\vary}\prop]{\argsep{#1}{\mathrel{\comp{=}}}}
```

We refer to [KM] for the details of the notation specification; suffice it to say that this allows us to write \eq{\varx,\vary,\varz} to obtain $x = y = z$.

A natural next step is to then also allow for *variable sequences* of unspecified length, which we can declare in sTeX via
Updated 2023-04-25: syntax \varseq{seqx}[type=\setA]{1,\ellipses,\svar{n}}{{\comp x}_{#1}},
giving us the semantic macro \seqx. \seqx{i} then produces x_i, \seqx! produces $x_1, \ldots, x_n$, and \seqx can be used as argument for any flexary argument in a semantic macro – e.g. \eq\seqx yields $x_1 = \ldots = x_n$.

This poses the question how we translate all these representations into formal Mmt/OMDoc, where equality is supposed to have type $(x \in A, y \in A) \to$Prop. In a first step, Mmt wraps the argument sequence into an application of a dedicated constant seq(x,y,z), to clearly distinguish the boundaries of the argument sequence from (potential) neighbouring arguments. Additionally, we provide typical operations on sequences as constants, such as map, fold, head/tail etc. with corresponding simplification and typing rules. Lastly, we allow authors to specify the precise behaviour of a flexary operator via an additional key assoc= for \symdecl with the following possible values:

pre	individually prefixed, e.g. $\forall x, y, z.P$ resolving to $\forall x.\forall y.\forall z.P$.
binr	right-associative binary, e.g. $A \to B \to C$ resolving to $A \to (B \to C)$
binl	analogously left-associative,
bin	associative binary,
conj	conjunctive[2] binary, as in $A = B = C$ resolving to $A = C \wedge B = C$.

Mmt attaches this information as meta data to the declaration, and subsequently uses it to insert (mainly) folds over the provided sequences in symbol applications to guarantee that typing checks succeed correctly. In the case where a sequence is *definite* (rather than variable), simplification rules on the sequence operations yield the actual desired outcome (e.g. $x = y = z$ resolving to $x = z \wedge y = z$).

3.4 Statements (Theorems, Definitions, Axioms, Proofs, ...)

Statements are among the most central notions in mathematics and should be adequately represented in sTeX. But they are also the representational feature, where informal and formal practice differ most. In most formal systems, statements are represented as compact expressions, whereas in informal mathematics, they are usually type-labeled and (often) numbered text fragments (sometimes multiple paragraphs long) with (explicit or implicit) premises, may contain (sub-)definitions or lemmata, and in general contain a lot more structure than mere closed formal expressions. A single informal statement usually corresponds to multiple formal ones. Thus sTeX needs to allow in-situ markup for the statement fragments in a way that this correspondence is made explicit without disrupting the (PDF) presentation.

[2] This case requires a symbol for conjunction to be in scope, which can be marked as such using a dedicated parametric Mmt rule. The expression $x = y = z$ above (and now here, too) type checks, because we included a module that provides one earlier.

We support doing so by allowing semantic macros outside of math mode to annotate arbitrary text, mark variable declarations inside statements as universally bound, and explicitly markup the *definiens* or *conclusion* of a statement. Similar support for *premises* and other statement components is planned. The MMT import is subsequently tasked with assembling all these individual components of a statement into the single (intended) formal expression.

For example, we can assert the reflexivity for our equality thusly:

```
\begin{sassertion}[style=axiom,name=equality-reflexive]
  \conclusion{\foral{The \symref{equal}{equation}
    $\arg[2]{\eq{\varx,\varx}}$ holds
    \comp{for all} $\arg[1]{\inset\varx\setA}$}}.
\end{sassertion}
```

yielding:

Axiom 1. The equation $x = x$ holds for all $x \in A$.

Again, we refer to [KM] for syntactic details, and merely note that we use the already present semantic macros to annotate the statement, which is also marked-up as one single *conclusion* with no premises. For a more elaborate example, see Definition 2.

The *judgments-as-types* paradigm has been proven to be effective to model declaratively true propositions, axiomatic (via undefined declarations) or proven (via defined declarations), both in general as well as in MMT specifically. Within STEX, this is reflected by \begin{sassertion}[name=equality-reflexive] generating a symbol `equality-reflexive` with no semantic macro. MMT subsequently uses the contents of the \conclusion{}-macro and the semantic macros therein to construct the actual proposition \foral{\inset\varx\setA}{\eq{\varx,\varx}} (i.e. $\forall x \in A.x = x$) to assign `equality-reflexive` the type $\vdash (\forall x \in A.x = x)$, where judgmentholds maps propositions to types.

Definitions work similarly, except here, we do not generate a new constant. Instead, for each \definiendum[symbolname]{...}, MMT either attaches a definiens to `symbolname`, if `symbolname` was declared in the same module *and* was previously undefined, or generates an equality rule for the symbol and the new definiens otherwise.

Variables declared *within* a statement and marked with bind=forall are appropriately bound/abstracted away, as are previously declared variables marked with \varbind. In preparation for the next sections, this allows us to e.g. define associativity as a property of binary operations like this:

```
\vardef{varop}[args=a,op=\circ,assoc=bin,type=\funspace{\setA,\setA}\setA]
    {\argsep{#1}{\mathbin{\comp\circ}}}
\symdecl{associative}[args=1,type=\bind{\varop!}\prop]

\begin{sdefinition}[for=associative]
  \varbind{varop}

  An operation $\fun{\varop!}{\setA,\setA}\setA$ is called
```

```
\definame{associative}, if \definiens[associative]{\foral{$\arg[2]{
  \eq{
    \varop{(\varop{\varx,\vary}),\varz},
    \varop{\varx,(\varop{\vary,\varz})}
  }
}$ \comp{for all} $\arg[1]{\inset{\varx,\vary,\varz}\setA}$}}.
\end{sdefinition}
```

Yielding:

Definition 1. *An operation* $\circ : A \times A \to A$ *is called* ***associative****, if* $(x \circ y) \circ z = x \circ (y \circ z)$ *for all* $x, y, z \in A$.

MMT consequently generates the definiens $\{(A \in \mathtt{SET})\}_I(\circ \in A \times A \to A) \to \forall x, y, z \in A.(x \circ y) \circ z = x \circ (y \circ z)$ for the constant `associative`.

Notably, the sdefinition environment's for= key allows for a comma-separated list of multiple symbols, and may contain several \definiens-calls to allow for defining multiple symbols in the same statement. Additionally, every \definame, \definiendum and \definiens adds the referenced symbol to the for=-list automatically, and in the case where it contains a single element, the option [associative] for \definiens is optional, so the above example is redundant in several places. Also, note that sdefinition defers to the definition-environment provided by (in this paper) the `llncs` document class for the actual typesetting (this behaviour can me customized).

4 Implementing Structural Features in sTeX

Having investigated how to translate informal phenomena to MMT/OMDoc, we will now discuss the reverse direction: Representing the language features available in MMT in sTeX.

MMT implements a mechanism for generic structural features [CICM20]. In MMT surface syntax,

featurename name ((parameters)∗) = (declarations)∗ ❙

behaves *internally* like an MMT theory, but generates a derived declaration with name name instead. After closing the declaration (with ❙), MMT will look for a structural feature (a special rule implemented in Scala) with name featurename, that processes the parameters and the body of the declaration and elaborates it into a set of primitive declarations computed from the derived declaration.

If we want to allow for structural features in sTeX, we consequently need to mimic the elaboration behaviour of the respective feature within LaTeX, making a certain amount of code duplication unavoidable. Luckily, since LaTeX does not need to be aware of the precise semantics of declarations in general, the elaboration in sTeX only needs to cover the names and notations of elaborated symbols, not the (computationally considerably more complicated) types and definientia.

More importantly however, structural features are an *extension principle* of MMT – users can in principle provide new features anywhere. Consequently, STEX needs to be similarly extensible. Where structural features in MMT need to be specified in Scala, for STEX, their elaboration naturally needs to be specified using LATEX, and where MMT provides an *abstract class* for this purpose, STEX provides a dedicated environment structural_feature_module that takes care of much of the boilerplate.

The rest of this section will now cover the most important and (in some cases) *primitive* structural features of MMT.

4.1 Mathematical Structures (Module types)

Module types were introduced in [MRK18], already for the explicit purpose of mimicking the way mathematical structures – such as algebraic structures, topological spaces, ordered sets, and other models of a (usually first- or second-order) theory – are treated in informal mathematics.

Given a theory M, the type $\texttt{MOD}(M)$ represents a dependent record type with manifest fields, with a field of name d for every declaration with name d in M. Given a record $m : \texttt{MOD}(M)$, a field d is accessible via the record projection $m.d$. We refer to [MRK18] for the implementation details in MMT.

As an introductory example, the following code introduces the structured type of semilattices:

```
\begin{mathstructure}{semilattice}
  \symdef{universe}[type=\collection]{\comp U}
  \symdef{op}[type=\funspace{\universe,\universe}\universe,args=a,op=\circ,
    assoc=bin]{\argsep{#1}{\mathbin{\comp\circ}}}
  \begin{sdefinition}[for=semilattice]
      A structure $\mathstruct{\universe,\op!}$
      with $\fun{\op!}{\universe,\universe}\universe$
      is called a \definame{semilattice}, if
      \inlineass[name=associative-axiom]{\conclusion{
        \associative{$\arg{\op!}$ is \comp{associative}}%
      }},
      \inlineass[name=commutative-axiom]{\conclusion{
        \commutative{$\arg*{\op!}$\comp{commutative}}%
      }} and
      \inlineass[name=idempotent-axiom]{\conclusion{
        \idempotent{$\arg*{\op!}$\comp{idempotent}}%
      }}
  \end{sdefinition}
\end{mathstructure}
```

This generates a new (nested) module `semi-lattice-structure`, with constants `universe`, `op`, and the three axioms (the command \inlineass behaves like the sassertion-environment), and generates the following output:

Definition 2. *A structure $\langle L, \circ\rangle$ with $\circ : L \times L \to L$ is called a* ***semilattice****, if $\circ$ is associative, commutative and idempotent*

Since `semilattice` is defined as a structure with a sequence of axioms rather than a simple (object-level) definiens, we do not use the \definiens-macro here.

mathstructure also generates a new symbol `semilattice`. Subsequently, it allows us to *instantiate* `semilattice` via Updated 2023-04-25: functionality now subsumed by semdef/symvar \symdef or \vardef, after which we can access the individual fields of the instance by their names. For example, the code Updated 2023-04-25: changed

```
1  \vardef{varL}[return=\semilattice]{\mathcal{L}}
2  Let $\defeq{\varL!}{\mathstruct{\varL{universe},\varL{op}!}}$ a
3  \symname{semilattice}, then $\varL!$ is
4  \varL{commutative-axiom}{\comp{commutative}} by definition.
```

yields the following output:

Let $\mathcal{L} := \langle L_{\mathcal{L}}, \circ_{\mathcal{L}}\rangle$ a semilattice, then $\mathcal{L}$ is *commutative* by definition.

Additionally, we can provide explicit definientia for some of the fields in an instantiation:

```
1  \vardef{varLb}[return={\semilattice[universe=\setA]}]{\mathcal{L}_2}
2  Let $\defeq{\varLb!}{\mathstruct{\varLb{universe},\varLb{op}!}}$
3  a \symname{semilattice} on $\setA$.
```

Let $\mathcal{L}_2 := \langle L_{\mathcal{L}_2}, \circ_{\mathcal{L}_2}\rangle$ a semilattice on A.

Internally, the mathstructure delegates to the generic environment structural_feature_module, which returns the list of the MMT URIs of all symbols declared in or imported into its body. mathstructure stores this information in a dedicated LaTeX3 property list indexed by the name.

\symdef and \symvar then define a new property list the same way, but storing the respective code to invoke the assignment target, depending on whether it is a variable or a symbol, rather than the MMT URI. While this forbids assigning fields in an instantiation to arbitrary expressions, this is necessary to remain compatible with e.g. symbol references and notation usages. In practice, this limitation can be remedied by introducing a defined variable with the desired expressions as definiens. This also means, that \varLb{universe} does in fact expand to \setA directly.

For the unassigned fields, STEX instead generates new variables with names <instancename>.<fieldname>, copying the notations of the original declarations in the mathstructure-environment. For example, \varLb{op} ultimately expands to \stex_invoke_variable{varLb.op}, which inherits its notation (and all other required properties) from the \symdef{op} in semilattice-structure. The MMT importer is subsequently able to reconstruct the actual term varLb.op from the variable name for unassigned fields, and is provided with the simplification of the field projection already when using assigned fields.

4.2 Includes with Modification (MMT-structures)

In MMT, the name *structure* refers to a (usually named) include with modification – a theory morphism that copies the declarations of its domain into the codomain.

The contents of an MMT structure are internally represented with the same data structures as the declarations in a theory, and the surface syntax is identical in both cases; MMT however subsequently checks that every constant in a structure corresponds (by name) to one in the domain, and its components are interpreted as reassignments of those of the domain constant. The structure can thus assign new definientia, notations or aliases, provided judgments are preserved. Given a declaration `d` in a theory $\texttt{ns}_1\texttt{?T}$, its target in a structure `s` from some theory $\texttt{T}_1$ (including `T`) to $\texttt{ns}_2\texttt{?T}_2$ is given the MMT URI $\texttt{ns}_2\texttt{?T}_2\texttt{?s/[ns}_1\texttt{?T]/d}$. Notably however, in MMT's surface syntax, the name of the generated MMT URI can not effectively be expressed (due to the full URI $\texttt{ns}_1\texttt{?T}$ occurring in its name), making it in practice impossible to refer to this declaration without using either its notation or an alias – it does, however, make it possible to disambiguate between multiple declarations with the same name but different full MMT URIs.

Since the name "structure" strongly conflicts with the usage of the word in informal mathematics, we instead refer to the corresponding STEX feature as a copymodule. We also simplify the feature slightly by assuming that (as validated in practice) name clashes within one module (that is a meaningful domain of a copymodule) do not occur, and simply generate the name `?s/d` instead.

Similarly, since reusing the \symdecl macro would be syntactically misleading with respect to its semantics, we instead introduce dedicated macros \renamedecl and \assign to provide aliases and new definientia in a copymodule.

For \assign{symbolname}{term}, STEX merely resolves symbolname to its full MMT URI and exports the pair to the XHTML. \renamedecl{symbolname}{newname} generates a new semantic macro *\newname* for s/symbolname; the variant \renamedecl{symbolname}[anothername]{newname} additionally renames the symbol to anothername and is interpreted as an alias in the MMT import.

As a consequence, we can use copymodule environments in STEX exactly like MMT structures, for example to assemble a module of lattices via combining two distinct copies of semilattice and providing appropriate new notations:

Updated 2023-04-25: Syntax

```
1  \begin{mathstructure}{lattice}
2    \begin{copymodule}{joinsl}{semilattice}
3        \renamedecl{universe}[universe]{universe}
4        % the target of "universe" is now called "lattice—module?universe"
5        % rather than "lattice—module?joinsl/universe"
6        \renamedecl{op}[join]{join}
7    \end{copymodule}
8    \begin{copymodule}{meetsl}{semilattice}
```

```
9         \assign{universe}{\symname{lattice-module?universe}}
10        \renamedecl{op}[meet]{meet}
11    \end{copymodule}
12    \notation*{join}[op=\vee,vee]{\argsep{#1}{\mathbin{\comp\vee}}}
13    \notation*{meet}[op=\wedge,wedge]{\argsep{#1}{\mathbin{\comp\wedge}}}
```

We can then add the additional axioms using the techniques described above:

Definition 3. *A structure $\langle L, \vee, \wedge\rangle$ is called a **lattice**, if both $\langle L, \vee\rangle$ and $\langle L, \wedge\rangle$ are semilattices, and the equations $a \vee (a \wedge b) = a$ and $a \wedge (a \vee b) = a$ hold for all $a, b \in L$.*

*The operations $\vee$ and $\wedge$ are called **join** and **meet**, respectively.*

4.3 Parametric Theories, Views, Realizations and Realms

There are some special cases of MMT structures relevant for the following development:

1. *Unnamed* MMT structures are considered *implicit* – the targets of the individual declarations in the domain theory retain (and are referred to by) their original MMT URIs. In fact, MMT includes are internally represented simply as unnamed structures with no assignments. MMT requires however, that there is at most one implicit morphism between any two theories (which includes their compositions), and that the sub-diagram of theories with implicit morphisms is acyclic.
2. *Total* MMT structures are those that explicitly assign all declarations in their domain to some valid term over the codomain. Structures marked as total are checked by MMT with respect to this requirement.
3. Structures that are both total and unnamed are referred to as *realizations* and are treated in a special manner: If s is a realization from S to T, S' has an implicit morphism m from S, and T' has an implicit morphism from T, then whenever a new implicit morphism from S to T occurs, it is replaced by the corresponding *pushout* along m and s.

While views, as the most general theory morphisms, are conceptually fundamental to the MMT/OMDoc core design and ontology, in practice they are never actually *used* in the MMT surface language – instead, they have turned out to be always subsumed by a more appropriate kind of MMT structure; either a total structure or a realization.

Similarly, while MMT offers parametric theories and includes, the fact that an include of a parametric theory requires instantiating its parameters and is an implicit morphism implies that only one set of assignments for theory parameters can be supplied in the cone of outgoing implicit morphisms of any theory that includes a parametric theory.

What this means is that both views and parametric theories are fully subsumed by total structures and realizations for views, and realizations for (includes of) parametric theories. Consequently, in sTeX3 we decided to drop both

entirely in favor of total structures (which we call interpretmodule in sTeX) and realizations.

Realizations also allow us to implement *realms* [CFK14], at least in the special case where a canonical candidate for the *face* of the realm exists – a pillar, then, is any codomain of a realization with the face as its domain. In particular, realizations likely allow us to introduce *fully formal* foundations in sTeX– or, rather, they allow for developing modules using the generic (informal) symbols in a natural way, while obtaining their counterparts in some formal setting via the induced pushout along realizations from informal modules into formal ones, whenever such a realization is possible.

The syntax of the interpretmodule and realization environments is perfectly analogous to that of copymodule, with the only difference being that interpretmodule explicitly checks that every declaration in the domain is being assigned, and realization explicitly modifies the symbols of the domain and adds the Mmt URI of the domain to the list of imported modules. Since sTeX already checks that no previously imported module is "activated" twice (for efficiency reasons), this is sufficient to mimic the behaviour of an Mmt realization in the informal setting of LaTeX, where the precise types and definientia of symbol are semantically inaccessible and hence irrelevant anyway.

As a consequence, we can use interpretmodule to e.g. implement *endomorphisms* of modules, such as the *dual* of a lattice:

Updated 2023-04-25: syntax

```
\begin{smodule}{DualLattice}
  \importmodule{MMTStructures/lattice-module}
  \begin{interpretmodule}{dual}{lattice}
      \assign{universe}{\symname{lattice-module?universe}}
      \assign{join}{\meet!} \assign{meet}{\join!}
      \assign{joinsl/associative-axiom}{\symname{meetsl/associative-axiom}}
      ...
\end{interpretmodule} \end{smodule}
```

And we can use a realization to represent the canonical (i.e. implicit in the Mmt sense) isomorphism between the algebraic and order theoretic definitions of a lattice. For clarity, we will call the latter a *locally-bounded partial order*:

Definition 4. *A structure $\langle S, \leq\rangle$ is called a* ***locally-bounded partial order****, if $\leq$ is a partial order and for all $a, b \in S$ there is a* ***least-upper-bound*** $\texttt{lup}(a, b) \in S$ *and a* ***greatest-lower-bound*** $\texttt{glb}(a, b) \in S$*.*

We can then easily obtain the desired isomorphism like this: Updated 2023-04-25: Syntax

```
\begin{sassertion}[style=theorem]
  \begin{realization}{lattice}
    \assign{universe}{\symname{lborder-module?universe}}
    \assign{join}{\lup!} \assign{meet}{\glb!}
    \assign{joinsl/associative-axiom}{trivial}
    \assign{joinsl/commutative-axiom}{trivial}
```

```
      \assign{joinsl/idempotent-axiom}{trivial}
      \assign{meetsl/associative-axiom}{trivial}
      \assign{meetsl/commutative-axiom}{trivial}
      \assign{meetsl/idempotent-axiom}{trivial}
      \assign{absorp1-axiom}{trivial}
      \assign{absorp2-axiom}{trivial}
    \end{realization}
    Every \symref{lbporder}{locally-bounded partial order}
    $\mathstruct{\universe,\porder!}$ is a \symname{lattice}
    $\mathstruct{\universe,\join!,\meet!}$
    with $\eq{\join{\vara,\varb},\lup{\vara,\varb}}$ and
    $\eq{\meet{\vara,\varb},\glb{\vara,\varb}}$
  \end{sassertion}
```

Theorem 1. *Every locally-bounded partial order* $\langle L, \leq \rangle$ *is a lattice* $\langle L, \vee, \wedge \rangle$ *with* $a \vee b = \texttt{lup}(a, b)$ *and* $a \wedge b = \texttt{glb}(a, b)$

5 Conclusion & Future Work

We have presented our approach to integrating the sTeX package with the Mmt system. As a result, we are now able to use the full suite of formal representation mechanisms of Mmt/OMDoc from within LaTeX, without sacrificing the latter's typesetting capabilities. Notably, while "properly" specifying fully formal content in sTeX requires some expertise to a similar degree as other formal languages, subsequently *using* this content in a *seemingly* informal manner is no more difficult than most other LaTeX macros. Additionally, sTeX's module system allows for distributed and collaborative development of interconnected libraries.

A Formal Foundation for Informal Mathematics: Naturally, in order to apply useful services enabled by formal methods to informal mathematical documents (such as type checking) requires *some* formal foundation in terms of which the knowledge needs to be represented. As the examples in this paper demonstrate, we have already started experimenting with and implementing one such foundation by providing typing rules for some of the various symbols imported here. This poses the obvious question, what "*the*" adequate *formal foundation for informal mathematics* is, or rather, could be – or, more precisely, which typing and inference rules are necessary as primitives to support formal semantic services for mathematical concepts as defined and used in informal practice.

We are far from answering this question, if indeed a satisfying answer is even possible. Indeed, we conjecture that any plausible candidate for such a "foundation" is necessarily incomplete, context-dependent and likely even locally inconsistent , by virtue of informal practice often ignoring or glossing over technical aspects and details that distract from the matter at hand, but are technically needed for being formally rigorous.

However, we are now in a situation where we can empirically investigate this question by iteratively semantically annotating purely informal mathematical developments, and conveniently and easily experiment with plausible candidates by quickly providing ad-hoc rules where desirable.

References

[CFK14] Jacques Carette, William Farmer, and Michael Kohlhase. "Realms: A Structure for Consolidating Knowledge about Mathematical Theories". In: [Wat+14], pp. 252–266. URL: https://kwarc.info/kohlhase/papers/cicm14-realms.pdf.

[CICM20] Dennis Müller et al. "Representing Structural Language Features in Formal Meta-Languages". In: *Intelligent Computer Mathematics (CICM) 2020*. Ed. by Christoph Benzmüller and Bruce Miller. Vol. 12236. LNAI. Springer, 2020, pp. 206–221. URL: https://kwarc.info/kohlhase/papers/cicm20-features.pdf.

[HKR14] Fulya Horozal, Michael Kohlhase, and Florian Rabe. "Flexary Operators for Formalized Mathematics". In: [Wat+14], pp. 312–327. URL: https://kwarc.info/kohlhase/papers/cicm14-lfs.pdf.

[KM] Michael Kohlhase and Dennis Müller. *The sTeX3 Package Collection*. Tech. rep. URL: https://github.com/slatex/sTeX/blob/main/doc/stex-doc.pdf (visited on 04/24/2022).

[Koh06] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://kwarc.info/kohlhase/papers/mcs08-stex.pdf.

[MMT] *MMT – Language and System for the Uniform Representation of Knowledge*. Project web site. URL: https://uniformal.github.io/ (visited on 01/15/2019).

[MRK18] D. Müller, F. Rabe, and M. Kohlhase. "Theories as Types". In: *Automated Reasoning*. Ed. by D. Galmiche, S. Schulz, and R. Sebastiani. Springer, 2018, pp. 575–590.

[Mül19] Dennis Müller. "Mathematical Knowledge Management Across Formal Libraries". PhD thesis. Informatics, FAU Erlangen-Nürnberg, Dec. 2019. URL: https://opus4.kobv.de/opus4-fau/files/12359/thesis.pdf.

[RK13] Florian Rabe and Michael Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: https://kwarc.info/frabe/Research/mmt.pdf.

[sTeX] *sTeX: A semantic Extension of TeX/LaTeX*. URL: https://github.com/sLaTeX/sTeX (visited on 05/11/2020).

[Wat+14] Stephan Watt et al., eds. *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. LNCS 8543. Springer, 2014.