MMT Internals
An Ongoing Tutorial

Florian Rabe, Jonas Betzendahl
(Illustrations by Katja Berčič)

February 28, 2019

Contents

1 Introduction

2 Overview (2018-11-28/29)

Full session on YouTube: Part 1, Part 2.

2.1 Structural Elements and Objects

Further reading: Structure

All MMT content is divided into the *structure level* and the *object level* (compare Figure 1). The structure level is a tree of so-called structural elements (see the corresponding class, `StructuralElement`) that all have a path (*D*ocuments have a `DPath`, *M*odules have an `MPath` and Declarations have a `GlobalName`; the latter two also have a name, which is a `LocalName`).
The general idea is that documents (everything from a repository to a directory to a file to a section within a file can be viewed as a document here) contain lists of modules and modules contain lists of declarations. Both modules and documents can also contain other modules or documents respectively. Narrative structure of documents (what files are in what directories etc.) does not carry any semantics. For this, modules should be used instead.

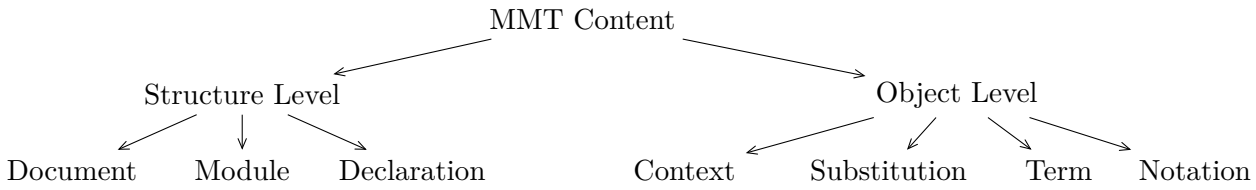

Figure 1: Overview of content structure. Nodes in this tree generally correspond to Scala classes.

The prominent examples for modules would be theories or views and for declarations constants. These in terms have components that are objects. For example, theories have metatheories, views have domains and codomains, and constants have any number of types, definition and notations, which are all *terms*.
The following is a good first overview about what forms MMT terms take (for more on this, also see Section 3, where the object level is discussed in more detail):

- **OMS**
  (OpenMath Symbol, refers to a constant)
- **OMA**
  (OpenMath Application, takes operator/function and children/arguments)
- **OMBIND**
  (OpenMath Binder, binds variables)
- **OMV**
  (OpenMath Variable)
- **OMLIT**
  (OpenMath Literals)

2.2 Algorithms

MMT's architecture has three phases. These correspond most interesting and relevant algorithms MMT offers, which will we be discussing next. The three phases are:

- **Lexing / Parsing**
  (no real distinction, we'll refer to this as just "parsing" from now on)

- **Type Checking**
  (this includes type inference and type reconstruction)
- **Simplification**
  (this also includes elaboration)

Each of these phases is called on each declaration separately and the first declaration is entirely processed (through all three phases) before the second declaration is touched. It is *not* the case that MMT first parses everything, then typechecks everything and finally simplifies everything. This is so because the successful processing of a later declaration might depend on the complete result of an earlier declaration (imagine a notation being used that was only introduced earlier in the file).

The code equivalent to these algorithms are also separated along the same divide of object- and structure-level. So the MMT API contains Scala classes like `StructureParser`, `StructureChecker` and `StructureSimplifier` as well as `ObjectParser`, `ObjectChecker` and `ObjectSimplifier`.
`StructureX`s take `ObjectX`s as arguments, to ensure modularity. For example, you could probably write an entirely new StructureParser in an afternoon, that you could then use with the already existing object parser without any problems.
Any IDE features are built on top of this, meaning that any new component here could also easily be used for development in IDEs.

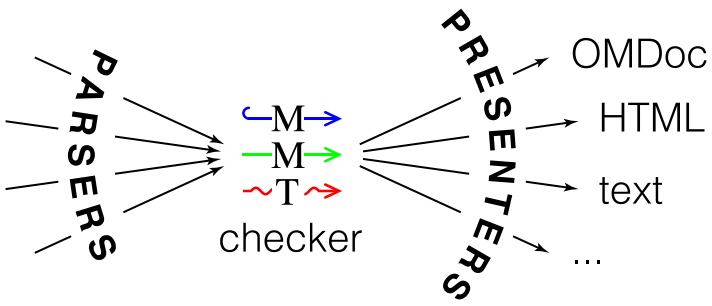

Figure 2: There can be many parsers and many presenters, but they all use the same MMT checker

As a general rule, the pipeline flow is many to one to many (compare Figure 2). There are multiple parsers and presenters (like HTML, plaintext, ...) and you could relatively easily add your own (if you would like a parser that is more suitable to your particular needs, like, say, one that parses JSON). However, every instantiation of the pipeline uses the same MMT checker.
The standard classes for each of these is given in the table in Figure 3.

	Structure:	**Object**:
Parsing:	`KeywordBasedParser`	`NotationBasedParser`
Checking:	`MMTStructureChecker`	`RuleBasedChecker`
Simplifying:	`ElaborationBasedSimplifier`	`RuleBasedSimplifier`

Figure 3: Standard Names

In the following sections, we will take a closer look at each of the three main phases.

2.2.1 Parsing

Further reading: Delimiters

Parsing in MMT is usually (if you don't roll your own differently) done with delimiters and keyword. Everything starts with a keyword and ends with a delimiter. If you've written some MMT surface syntax, you will be already familiar with these delimiters.

This structure can also easily be nested.
The most important delimiters (not bothering with document delimiters) are:

- `MD` (Module Delimiter)
- `DD` (Declaration Delimiter)
- `OD` (Object Delimiter)

There are two classes, `ParsingStream` and `ParsingUnit`, which encapsulate just about everything you conceivably would want to parse.

2.2.2 Type Checking

2.2.3 Simplification

3 Terms (2019-01-31)

Full session on YouTube: Link.

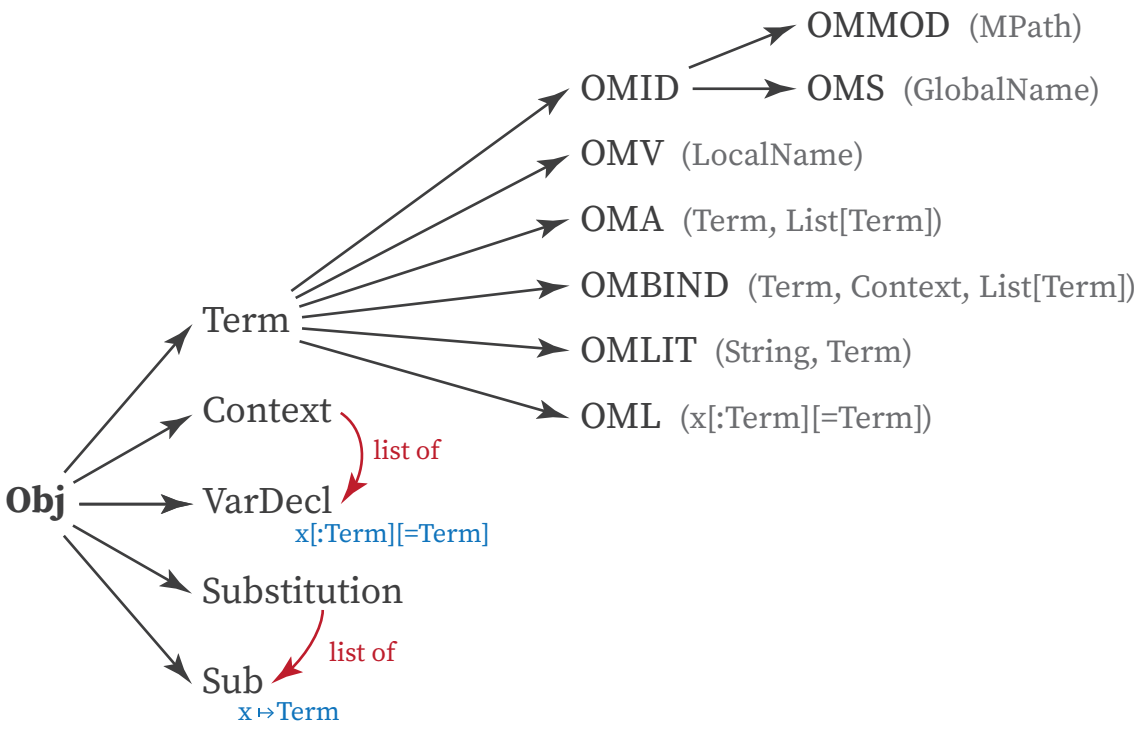

Figure 4: Overview of MMT Terms