MMT Internals
An Ongoing Tutorial

Florian Rabe, Jonas Betzendahl

February 27, 2019

Contents

1 Overview (2018-11-28/29)

1.1 Structure

Full session on YouTube: Part 1, Part 2.

Further reading: Structure

All MMT content is divided into the *structure level* and the *object level* (compare Figure 1.1). The structure level is a tree of named declarations that all have a path (*D*ocuments have a `DPath`, *M*odules have an `MPath` and Declarations have a `Globalname`). The general idea is that Documents contain lists of modules and modules contain lists of declarations. Both modules and documents can also contain other modules or documents respectively.
As an intuition for what these refer to in practice, one can think of modules as theories or views and of declarations as constants.

Narrative structure (what files are in what directories etc.) does not carry any semantics. For this, modules should be used instead.
The following is a good first overview about what forms MMT terms take (for more on this, also see Section 2, where the object level is discussed in more detail):

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

- **OMLIT**
  (OpenMath Literals)

1.2 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 the case 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. Any IDE features are built on top of this.

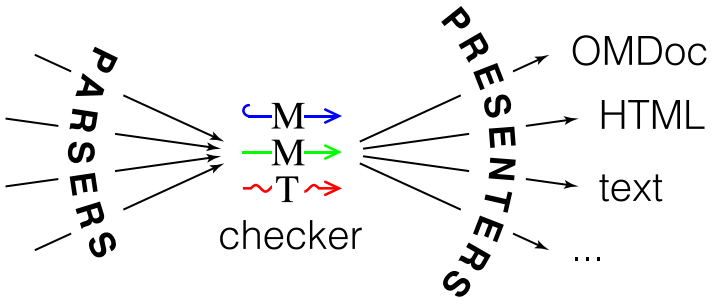

Figure 1: There can be many parsers and many presenters, but they all use the same MMT checker

As a general rule, the pipeline flow is many to one to many (compare Figure 1). There are multiple parsers and you could relatively easily add your own (if you would like a different presenter or parser that is more suitable to your particular needs). The same is true for presenters. However, every iteration of the pipeline uses the same MMT checker.
The standard classes for each of these is given in the table in Figure 2.

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

Figure 2: Standard Names

Should you want to write your own `StructureParser` or `ObjectSimplifier`, keep in mind that every parser subclasses `Importer` and every presenter subclasses `Exporter`.
In the following sections, we will take a closer look at each of the three main phases.

1.2.1 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.

1.2.2 Type Checking

1.2.3 Simplification

2 Terms (2019-01-31)

Full session on YouTube: Link.

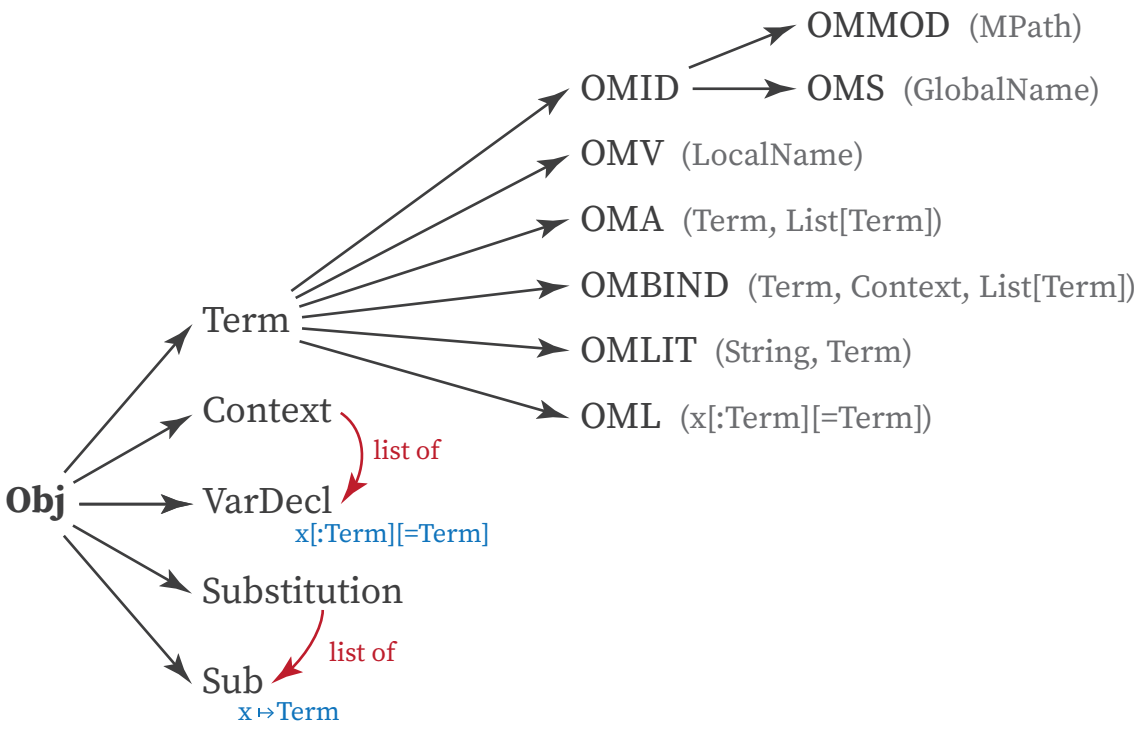

Figure 3: Overview of MMT Terms