The MMT Language

Florian Rabe

Jacobs University Bremen

Abstract. We introduce the MMT language, which provides a simple and scalable Module system for the development of Mathematical Theories. MMT permits to encode mathematical knowledge in a logic-neutral representation format that can represent the meta-theoretic foundations of mathematical and logical systems together with the represented knowledge itself and interlink the foundations at the meta-logical level. This "logics-as-theories" approach makes system behaviors as well as their represented knowledge interoperable and thus comparable.
Furthermore, MMT defines logic-independent notions of well-formedness and equivalence of modular mathematical theories. Thus, it provides an interface layer between formally rigorous mathematical systems, and knowledge management services which are not aware of the semantics of the processed knowledge.

1 Introduction

Mathematical knowledge is at the core of science, engineering, and economics, and we are seeing a trend towards employing computational systems like semi-automated theorem provers, model checkers, computer algebra systems, constraint solvers, or concept classifiers to deal with it. It is a characteristic feature of these systems that they either have mathematical knowledge implicitly encoded in their critical algorithms or (increasingly) manipulate explicit representations of the relevant mathematical knowledge often in the form of logical formulas. Unfortunately, these systems have differing domains of applications, foundational assumptions, and input languages, which makes them non-interoperable and difficult to compare and evaluate in practice. Moreover, the quantity of mathematical knowledge is growing faster than our ability to formalize and organize it, aggravating the problem that mathematical software systems cannot share knowledge representations.

The work reported in this paper focuses on developing an exchange format between math-knowledge based systems. We concentrate on a foundationally unconstrained framework for knowledge representation that allows to represent the meta-theoretic foundations of the mathematical knowledge in the same format and to interlink the foundations at the meta-logical level. In particular, the logical foundations of domain representations for the mathematical knowledge can be represented as modules themselves and can be interlinked via meta-morphisms. This "logics-as-theories" approach makes systems behavior as well as their represented knowledge interoperable and thus comparable at multiple

levels. The explicit representation of epistemic foundations also benefits systems whose mathematical knowledge is only implicitly embedded into the algorithms. Here, the explicit representation can serve as a documentation of the system interface as well as a basis for verification or testing attempts.

Of course communication by translation to the lowest common denominator logic – the current state of the art – is always possible. But such translations lose the very structural properties of the knowledge representation that drive computation and led to the choice of logical system in the first place. Therefore our format incorporates a module system geared to support flexible reuse of knowledge items via theory morphisms. This module system is the central part of the proposed format – emphasizing interoperability between theorem proving systems, and the exchange and reusability of mathematical facts across different systems. In contrast to the formula level, which needs to be ontologically unconstrained, the module system level must have a clear semantics (relative to the semantics of the formula level) and be expressive enough to encode current structuring practice so that systems can communicate without losing representational structure.

On a practical level, an exchange format must be *scalable* in the sense that it supports the distribution of resources (theories, conjectures, proofs, etc.) over the internet, so that they can be managed collaboratively. In the current web architecture this means that all (relevant) resources must be addressable by a URI-based naming scheme [2]. Note that in the presence of a complex modularity and reusability infrastructure, this may mean that resources have to be addressable, even though they are only virtually induced by the inheritance structure.

Finally the design, implementation, and maintenance of large scale logical knowledge management services will realistically only pay off if the same framework can be reused for different foundations of mathematics. Therefore, an interface layer is needed between the logical-mathematical core of a mathematical foundation and the needs of a foundation-independent knowledge management service that has to preserve the semantics of the knowledge it operates on without knowing it.

MMT was designed as such a representation language. It represents logical knowledge on three levels: the module, symbol, and object level. On the module level, we build on modular representation languages for logical knowledge such as OBJ [9], ASL [16], development graphs [1], and CASL [5]. In particular, we carry on the rigorous use of theories and theory morphism as the primitive modular concepts and the distinction between definitional and postulated links as the concepts relating theories. This distinguishes MMT from module systems that have been developed for type theories such as PVS [12], Isabelle [13], Coq [3], or Nuprl [6].

On the symbol level, MMT has in common with type theories the use of typed, possibly defined constants as the primitive concept. In particular, MMT uses the Curry-Howard correspondence ([7,10]) to represent axioms and theorem as constants, and proofs as terms. This distinguishes MMT from the logic-oriented module systems mentioned above.

Finally, on the object level, MMT uses the formal grammar of OPENMATH [4] to represent terms without committing to a specific formal foundation. Specific foundations are defined by defining judgments for typing and equality of terms, in which MMT is parametric. This distinguishes MMT from the logical and type theoretical languages above, which are either designed for one foundation (such as type theories) or pose heavyweight requirements on the foundation (such as being an institution in the case of CASL).

2 Syntax

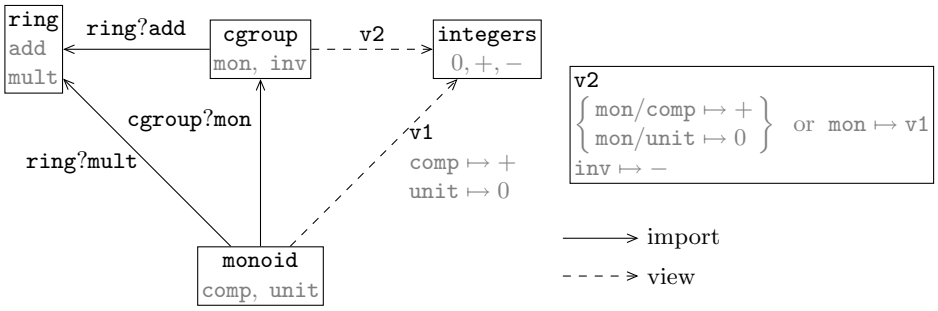


Fig. 1. Example

Example 1. We begin the exposition of our language with a simple motivating example that already shows some of the crucial features of MMT and that we will use as a running example: Fig. 1 gives a theory graph for a portion of the algebraic hierarchy. The bottom node represents the theory of monoids, which declares constants for composition and unit. (We omit the axioms and the types here.) The theory `cgroup` for commutative groups arises by importing from `monoid` and adding a constant `inv` for the inverse element. This theory does not have to declare constants for composition and unit again because they are imported from the theory of monoids. The import is named `mon`, and it can be referenced by the qualified name `cgroup?mon`; it induces a theory morphism from monoids to commutative groups.

Then the theory of rings can be formed by importing from `monoid` (via an import named `mult` that provides the multiplicative structure) and from `cgroup` (via an import named `add` that provides the additive structure). Because all imports are named, different import paths can be distinguished: By concatenating import and symbol names, the theory `ring` can access the symbols `ring?add/mon/comp` (addition), `ring?add/mon/unit` (zero), `ring?add/inv` (additive inverse), `ring?mult/comp` (multiplication), and `ring?mult/unit` (one).

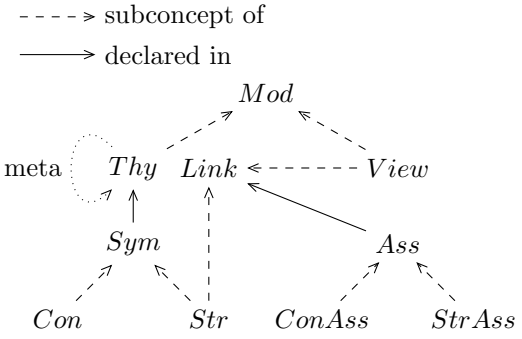
The node on the right side of the graph represents a theory for the integers declaring the constants 0, +, and −. The fact that the integers are a monoid is represented by the view `v1`. It is a theory morphism that is explicitly given by its interpretations of `comp` as + and of `unit` as 0. (If we did not omit axioms,

this view would also have to interpret all the axioms of `monoid` as – using Curry-Howard representation – proof terms.)

The view `v2` is particularly interesting because there are two ways to represent the fact that the integers are a commutative group. Firstly, all operations of `cgroup` can be interpreted as terms over `integers`: This means to interpret `inv` as $-$ and the two imported constants `mon/comp` and `mon/unit` as $+$ and 0, respectively. Secondly, `v2` can be constructed along the modular structure of `cgroup` and use the existing view `v1` to interpret all constants imported by `mon`. In MMT, this can be expressed elegantly by the interpretation $\texttt{mon} \mapsto \texttt{v1}$, which interprets a named import with a theory morphism. The intuition behind such an interpretation is that it makes the right triangle commute: `v2` is defined such that $\texttt{v2} \circ \texttt{cgroup?mon} = \texttt{v1}$. Clearly, both ways lead to the same theory morphism; the second one is conceptually more complex but eliminates redundancy. (This redundancy is especially harmful when axioms are considered, which must be interpreted as expensive-to-find proofs.)

Readers familiar with development graphs should note that the second variant of the view `v2` gives an example of a decomposition. In particular, the concepts of local and global links can be recovered as opposite extremes of MMT links.

Ontology and Grammar The central notion of MMT are **theory graphs**. They are directed multigraphs in which the nodes are theories and the edges are atomic theory morphisms. General theory morphisms are paths in the theory graph. We characterize theory graphs on three levels: the **module**, **symbol**, and **object** level. Modules and symbols are named structuring concepts, and objects are composed mathematical expressions. In the figure on the right, the relations between the structural MMT-concepts are defined in an ontology. Modules are either **theories** (Thy, e.g., `monoid` and `integers`) or **views** ($View$, e.g., `v1` and `v2`). Symbols are either **constants** (Con, e.g., `comp` and `inv`) or **structures** (Str, the imports from above, e.g., `mon` and `add`). On the symbol level, we also have **assignments** to symbols ($ConAss$, e.g., $\texttt{inv} \mapsto -$, and $StrAss$, e.g., $\texttt{mon} \mapsto \texttt{v2}$).



The concept of **links** ($Link$) arises as the union of structures and views. It captures their joint characteristics, namely that they induce theory morphisms and therefore occur as the edges in the theory graph. Structures being both symbols and links is a crucial feature of MMT that we will often exploit. Finally theory graphs are sets of modules, theories are sets of symbols, and links are sets of assignments to symbols. Syntactically, all three must be lists, but we will not permit repetitions and not make distinctions due to order.

The meta-variables we will use are given in Fig. 2. References to declared elements are Latin letters, objects and lists of elements are Greek letters. We will also use _ as an unnamed meta-variable for irrelevant values. The grammar for MMT is given in Fig. 3 where $^+$, |, and [−] denote non-empty repetition, alternative, and optional parts, respectively.

Web-scalability is built into MMT a priori: The productions for references end in the non-terminals URI and pchar, which are defined in RFC 3986. Thus, g produces URIs without queries or fragments (The query and fragment components of a URI are those starting with the special characters ? and #, respectively.), and pchar produces any Unicode character, possibly using percent-encoding for reserved characters. (Thus, percent-encoding is necessary for the characters ?/#[]% and all characters generally illegal in URIs.)

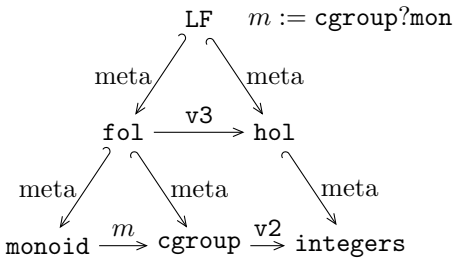
Level	Declaration	Expression
Module	theory T, S, R, M	theory graph γ (set of modules)
	view v	
	link m	
Symbol	constant c	theory body ϑ (set of symbols)
	structure r, s	link body σ (set of assignments)
Object	variable x	term ω
		morphism μ

Fig. 2. Meta-Variables

Theory graph	γ	::=	$\cdot \mid \gamma, Thy \mid \gamma, Viw$
Theory	Thy	::=	$T \stackrel{[M]}{:=} \{\vartheta\}$
View	Viw	::=	$v : S \to T \stackrel{[\mu]}{:=} \{\sigma\} \mid v : S \to T := \mu$
Theory body	ϑ	::=	$\cdot \mid \vartheta, Con \mid \vartheta, Str$
Constant	Con	::=	$c : \omega := \omega \mid c : \omega \mid c := \omega \mid c$
Structure	Str	::=	$s : S \stackrel{[\mu]}{:=} \{\sigma\} \mid s : S := \mu$
Link body	σ	::=	$\cdot \mid \sigma, ConAss \mid \sigma, StrAss$
Ass. to constant	$ConAss$	::=	$\vec{c} \mapsto \omega$
Ass. to structure	$StrAss$	::=	$\vec{s} \mapsto \mu$
Term	ω	::=	$\top \mid T?\vec{c} \mid x \mid \omega^{\mu} \mid @(\omega, \omega^+)$
			$\mid \beta(\omega; \Upsilon; \omega) \mid \alpha(\omega; \omega = \omega)$
Variable context	Υ	::=	$\cdot \mid \Upsilon, \omega$ if $str(\omega)$ of the form x
Morphism	μ	::=	$id_T \mid T?\vec{s} \mid v \mid \mu \bullet \mu$
Module reference	S, T, M, v	::=	$g?I$
Qualified identifier	$\vec{c}$	::=	$c \mid s/\vec{c}$
	$\vec{s}$	::=	$s \mid s/\vec{s}$
URI	g	::=	URI, no query, no fragment
Unqualified identifier	I, c, s, x	::=	pchar$^+$
	URI, pchar		see RFC 3986 [2]

Fig. 3. The Grammar for MMT Expressions

Meta-Theories The meta-relation is a binary relation between theories. The intuition is that the meta-theory M of T provides the syntactic material that is used to define the semantics of the symbols declared in T. For example, we can add meta-theories to Ex. 1 as in the figure on the right. `fol` is a theory for first-order logic, in which monoids and groups are axiomatized, `hol` is a theory for higher-order logic, in which the integers are defined. We can iterate this step and define both `fol` and `hol` using a logical framework like LF as a further meta-theory.



Because M is always included into T, any link with domain T must also provide assignments for the symbols of M. This is the role of the meta-morphism from M to the codomain of the link (or its meta-theory). For the link `cgroup?mon`, this is the identity $id_{\texttt{fol}}$ because domain and codomain have the same meta-theory. For the link `v2`, we need to give a meta-morphism `v3` from `fol` to `hol`.

Intuitive Semantics Among the modules, theory declarations are of the form $T \stackrel{M}{:=} \{\vartheta\}$ where T is the name, ϑ the body, and M an optional meta-theory. For views, there are two productions: $v : S \to T \stackrel{\mu}{:=} \{\sigma\}$ declares a view from the theory S to the theory T with name v, body σ and optional meta-morphism μ; $v : S \to T := \mu$ declares v as an abbreviation of the morphism μ.

Among the symbols, constants are of the form $c : \tau := \delta$ where c is the name and the terms τ and δ are optional type and definition. For structures, there are two productions that correspond to those for views: The only difference is that only the domain theory is given explicitly – the codomain is always the theory in which the structure is declared. Looking at the assignments, we encounter a crucial symmetry between the two classes of symbols and the two classes of objects: terms ω are to constants $\vec{c}$, as morphisms μ are to structures $\vec{s}$.

Denotationally, the intuitive semantics of an assignment $c \mapsto \omega$ in the body of a link m from S to T is that m maps the constant c declared in S to the term ω over T. Assignments of the form $c \mapsto \top$ are special: They express that the constant c is **hidden**, i.e., m is partial and undefined for c. Secondly, if r is a structure declared in S and μ a morphism over (i.e., into) T, then $r \mapsto \mu$ expresses that m maps r to μ. This means that the triangle $S?r \circ m = \mu$ commutes.

Operationally, the intuitive semantics of a structure s with domain S declared in T is that all symbols of S are copied into T. For example, if c is a constant of S, then s/c is the qualified name of a constant of T induced by s. In other words, the slash dereferences a structure. Similarly, if a link with domain T contains an assignment to s, this induces an assignment to the induced constant s/c. Furthermore, assignments may be **deep** in the following sense: If c is a constant of S, a link with domain T may also contain assignments to the induced constant s/c. In well-formed links, we will make sure there are no clashes, e.g., that a link does not contain assignments for both s and s/c.

Objects Finally we distinguish terms and morphisms at the object level of MMT. Terms are a generalization of OPENMATH objects: As such their productions contain constants $T?\vec{c}$ where $\vec{c}$ is a qualified constant name in the theory T; applications $@(\omega, \omega_1, \ldots, \omega_n)$ of ω to n arguments; bindings $\beta(\omega; \Upsilon; \omega')$ with binder ω, list of attributed variables Υ, and scope ω'; and attributions $\alpha(\omega; \omega_{key} = \omega_{value})$. We use the function $str(\omega)$ to remove toplevel attributions from a term. Finally ω^μ applies the morphism μ to the term ω.

Morphisms arise by composing links, i.e., qualified structure names $T?\vec{s}$ and views v, where $\mu \bullet \mu'$ is composition in diagrammatic order. We obtain an elegant symmetry between terms and morphisms if we think of morphisms from S to T as objects over T of type S, i.e., the codomain of a morphism corresponds to the "home theory" of a term, and its domain corresponds to the type of a term.

3 Foundation-Independent Semantics

3.1 Elaboration and Normalization

It turned out that the understanding of structures and their induced symbols is crucial to achieve foundation-independence. While this is easy for systems built for a particular foundation, it presents great difficulties for generic knowledge management services. In the following we give a very simple definition of the semantics of structures. Denotationally, it describes the data structures needed to interpret MMT theory graphs. Operationally, it is an **elaboration** of a theory graph into one without structures. We will first describe the judgments and functions and then give their mutually recursive definitions.

Firstly, we define the theories and links that comprise the theory graph described by γ. The judgment $\gamma \gg T = \{\vartheta\}$ expresses that T is a theory in T with body ϑ. We write $Thy(\gamma)$ for the set of references T for which this judgment holds, and in that case the function $\gamma(T)$ returns ϑ. Furthermore, we write $M \hookrightarrow T$ if T has meta-theory M.

$\gamma \gg m : S \to T := B$ expresses that m is a link from S to T in γ where B is of the form σ or μ according to whether m is defined by a link body or a morphism. We write $Link(\gamma)$ for the set of link references for which this judgment holds, and in that case the function $\gamma(T)$ returns (S, T, B). Furthermore, we write $\mu' \hookrightarrow m$ if m has meta-morphism μ'.

Secondly, for every theory or link of γ, we describe its components. For every $T \in Thy(\gamma)$, we use the judgment $\gamma \gg_T \vec{c} : \tau := \delta$ to express that $\vec{c} : \tau := \delta$ is an induced constant declaration of T. To avoid case distinctions, we permit the value $\bot$ for τ and δ if they are omitted. We write $Con^\gamma(T)$ for the set of constant identifiers $\vec{c}$ for which this judgment holds, and in that case the function $\gamma^T(\vec{c})$ returns (τ, δ).

For every $m \in Link(\gamma)$, we use the judgment $\gamma \gg_m \vec{c} \mapsto \delta$ to express that $\vec{c} \mapsto \delta$ is an induced assignment of T. To avoid case distinctions, we permit the value $\bot$ for δ if no assignment is provided. We write $Con^\gamma(m)$ for the set of constant identifiers $\vec{c}$ for which this judgment holds, and in that case the function $\gamma^m(\vec{c})$ returns δ.

Finally, the definitions of the above judgments are given in Fig. 4 and 5. To state the rules compactly, we define several shortcuts. $Mod \in \gamma$ expresses that a module declaration occurs in γ, and $Sym \in^\gamma T$ as well as $Ass \in^\gamma m$ express that a symbol or assignment occurs in a theory or link. When omitted types or definitions are translated along a morphism, they stay omitted, i.e., we put $\bot^\mu = \bot$. Furthermore, rules with square brackets abbreviate two rules: one with and one without the optional parts. Finally, we use rules with multiple conclusions to merge two rules that have the same premises.

$$\frac{T \overset{[M]}{:=} \{\vartheta\} \in \gamma}{\gamma \gg T = \{\vartheta\} \quad [M \hookrightarrow T]}$$

$$\frac{v : S \to T \overset{[\mu]}{:=} \{\sigma\} \in \gamma}{\gamma \gg v : S \to T := \sigma \quad [\mu \hookrightarrow v]} \qquad \frac{v : S \to T := \mu \in \gamma}{\gamma \gg v : S \to T := \mu}$$

$$\frac{s : S \overset{[\mu]}{:=} \{\sigma\} \in^\gamma T}{\gamma \gg T?s : S \to T := \sigma \quad [\mu \hookrightarrow v]} \qquad \frac{s : S := \mu \in^\gamma T}{\gamma \gg T?s : S \to T := \mu}$$

$$\frac{\gamma \gg T?s : S \to T := \_ \quad \gamma \gg S?\vec{r} : R \to S := \_}{\gamma \gg T?s/\vec{r} : R \to T := S?\vec{r} \bullet T?s}$$

Fig. 4. Induced Theories and Links

Definition 1. *A theory graph γ is called* clash-free *if $\gamma(-)$, $\gamma^T(-)$ for $T \in Thy(\gamma)$, and $\gamma^m(-)$ for $m \in Link(\gamma)$ are well-defined functions.*

Clash-freeness essentially guarantees the uniqueness of names. It is easily decidable. For example, a theory graph is ill-formed if the same module or constant name is declared twice. It is also ill-formed if the assignments in a link clash, e.g., if a link contains assignments for both s and s/c.

Normalization eliminates all morphism applications, expands all definitions, and enforces the strictness of hiding (A term with a hidden subterm is also hidden.). Thus, the normalization provides the MMT-specific part of the axiomatization of equality.

For a fixed theory graph γ, we write $\overline{\omega}$ for the normal form and define it by structural induction. The straightforward cases are:

$$\begin{aligned}
\overline{\top} &:= \top \\
\overline{x} &:= x \\
\overline{T?\vec{c}} &:= \begin{cases} \overline{\delta} & \text{if } \gamma \gg_T \vec{c} : _ := \delta,\ \delta \neq \bot \\ T?\vec{c} & \text{otherwise} \end{cases}
\end{aligned}$$

$$\frac{c : \tau := \delta \in^{\gamma} T}{\gamma \gg_T c : \tau := \delta}$$

$$\frac{\gamma \gg T?s : S \to T := \_ \quad \gamma \gg_S \vec{c} : \tau := \delta \quad \gamma \gg_{T?s} \vec{c} \mapsto \delta',\ \delta' \neq \bot}{\gamma \gg_T s/\vec{c} : \tau^{T?s} := \delta'}$$

$$\frac{\gamma \gg T?s : S \to T := \_ \quad \gamma \gg_S \vec{c} : \tau := \delta \quad \gamma \gg_{T?s} \vec{c} \mapsto \bot}{\gamma \gg_T s/\vec{c} : \tau^{T?s} := \delta^{T?s}}$$

$$\frac{\gamma \gg m : S \to T := \mu \quad \vec{c} \in Con^{\gamma}(S)}{\gamma \gg_m \vec{c} \mapsto (S?\vec{c})^{\mu}} \qquad \frac{\vec{c} \mapsto \omega \in^{\gamma} m}{\gamma \gg_m \vec{c} \mapsto \omega}$$

$$\frac{\vec{s} \mapsto \mu \in^{\gamma} m \quad \begin{array}{l}\gamma \gg m : S \to \_ := \_ \\ \gamma \gg S?\vec{s} : R \to \_ := \_\end{array} \quad \vec{c} \in Con^{\gamma}(R)}{\gamma \gg_m \vec{s}/\vec{c} \mapsto (R?\vec{c})^{\mu}}$$

Fig. 5. Induced Symbols and Assignments

$$\overline{@(\omega_1, \ldots, \omega_n)} := \begin{cases} @(\overline{\omega_1}, \ldots, \overline{\omega_n}) & \text{if } \overline{\omega_i} \neq \top \text{ for all } i \\ \top & \text{otherwise} \end{cases}$$

$$\overline{\beta(\omega_0; \omega_1, \ldots, \omega_n; \omega_{n+1})} := \begin{cases} \beta(\overline{\omega}; \overline{\omega_1}, \ldots, \overline{\omega_n}; \overline{\omega'}) & \text{if } \overline{\omega_i} \neq \top \text{ for all } i \\ \top & \text{otherwise} \end{cases}$$

$$\overline{\alpha(\omega_1; \omega_2 = \omega_3)} := \begin{cases} \alpha(\overline{\omega_1}; \overline{\omega_2} = \overline{\omega_3}) & \text{if } \overline{\omega_i} \neq \top \text{ for all } i \\ \top & \text{otherwise} \end{cases}$$

$$\overline{\omega^{id_T}} := \overline{\omega}$$

$$\overline{\omega^{\mu \bullet \mu'}} := \overline{\overline{\omega^{\mu}}^{\mu'}}$$

$$\overline{\top^m} := \top$$

$$\overline{x^m} := x$$

$$\overline{@(\omega_1, \ldots, \omega_n)^m} := \overline{@({\omega_1}^m, \ldots, {\omega_n}^m)}$$

$$\overline{\beta(\omega; \omega_1, \ldots, \omega_n; \omega')^m} := \overline{\beta(\omega^m; {\omega_1}^m, \ldots, {\omega_n}^m; {\omega'}^m)}$$

$$\overline{\alpha(\omega_1; \omega_2 = \omega_3)^m} := \overline{\alpha({\omega_1}^m; {\omega_2}^m = {\omega_3}^m)}$$

The only non-trivial case is the application of a link m to a constant $S?\vec{c}$. Firstly, if the domain of m is not S and $\mu \hookrightarrow m$, we put $\overline{(S?\vec{c})^m} := \overline{(S?\vec{c})^{\mu}}$. Otherwise, we assume $\gamma \gg_S \vec{c} : \_ := \delta$ and $\gamma \gg_m \vec{c} \mapsto \delta'$ and distinguish four cases:

$$\overline{(S?\vec{c})^m} := \begin{cases} \overline{\delta^m} & \text{if } \delta \neq \bot \\ \overline{\delta'} & \text{if } \delta = \bot,\ \delta' \neq \bot \\ \left\{ \begin{array}{ll} \overline{T?\vec{s}/\vec{c}} & \text{if } m = T?\vec{s} \text{ structure} \\ \top & \text{if } m = v \text{ view} \end{array} \right\} & \text{if } \delta = \delta' = \bot \end{cases}$$

If $S?\vec{c}$ has a definiens ($\delta \neq \bot$), it is expanded before applying m; firstly because m does not need to give assignments for defined constants, and secondly because m might hide the name $\vec{c}$. Otherwise, the assignment provided by m is used. If there is no definiens and m does not provide an assignment ($\delta = \delta' = \bot$), then m is a partial mapping, and there are two subcases: Partially defined structures map to the induced constant $T?\vec{s}/\vec{c}$; partially defined views hide their argument.

Since meta-theories can be considered as special structures, elaboration and normalization together permit to eliminate all MMT-specific constructs. Therefore, they can be used to define the semantics of γ:

Definition 2. *Two clash-free theory graphs γ and γ' are* semantically equivalent *if $Thy(\gamma) = Thy(\gamma')$ and for all $T \in Thy(\gamma)$ the functions $\gamma^T(-)$ and $\gamma'^T(-)$ agree up to normalization of their results, and correspondingly (ii) $Link(\gamma) = Link(\gamma')$ and for all $m \in Link(\gamma)$ the functions $\gamma^m(-)$ and $\gamma'^T(-)$ agree up to normalization of their results.*

This provides knowledge management services with a crucial invariant that permits them to represent and manipulate theory graphs foundation-independently. In particular, if γ is clash-free, we can replace (i) every structure s from S to T with the set of induced declarations $\gamma \gg_T s/\vec{c} : \tau := \delta$, and (ii) every assignment to that structure with the set of induced assignments $\gamma \gg_{T?s} \vec{c} \mapsto \delta$ without affecting the semantics of γ. The details can be found in [14].

3.2 Type System

Now we sketch the inference system that defines the **well-formed** theory graphs. The judgments are given in Fig. 6. Foundation-independence is achieved by making the inference system parametric in the judgments for typing and equality of terms. The definition of these judgments varies according to the foundation. We will discuss that in Sect. 4.

To avoid case distinctions, we also define typing judgments involving $\bot$. $\gamma \rhd_T \omega : \bot$ expresses that ω is an untyped value, and $\gamma \rhd_T \bot : \omega$ expresses that ω is a well-formed type.

Judgment	Intuition
$\rhd \gamma$	γ is a well-formed theory graph.
$\gamma \rhd \mu : S \to T$	μ is a well-typed morphism from S to T.
$\gamma \rhd_T \omega : \omega'$	ω is well-typed with type ω' over γ and T.
$\gamma \rhd_T \omega \equiv \omega'$	ω and ω' are equal over γ and T.

Fig. 6. Judgments

The inference rules for the structural levels are organized in such a way that we start with an empty theory graph and successively add three kinds of declarations:

– add a module at the end of γ,
– add a symbol at the end of the last module of γ (which must be a theory),
– add an assignment to a symbol to the last link of γ (which may be a view if γ ends in that view, or a structure if γ ends in a theory which ends in that structure).

When theories or links are added, their body must be empty initially and populated by subsequently added symbols or assignments. This has the effect that there is one rule for every production of the grammar.

We give all inference rules in the appendix and limit ourselves to some examples here. The following rule adds a view from S to T to a theory graph. If S has a meta-theory M, the view must include a meta-morphism μ from M to T.

$$\frac{\rhd \gamma \quad S \in \mathit{Thy}(\gamma) \quad T \in \mathit{Thy}(\gamma) \quad M \hookrightarrow S \quad \gamma \rhd \mu : M \to T}{\rhd \gamma,\ v\ :\ S \to T\ :\overset{\mu}{=}\ \{\cdot\}}\ \mathit{View}$$

The view is initially empty. If we put $\gamma' = \gamma,\ v\ :\ S \to T\ :\overset{\mu}{=}\ \{\sigma\}$, the following rule adds an assignment to it.

$$\frac{\rhd \gamma' \quad \gamma' \gg_S \vec{c}\ :\ \tau\ :=\ \delta \quad \gamma' \rhd_T \delta' : \tau^v \quad \gamma' \rhd_T \delta^v\ \leq\ \delta'}{\rhd \gamma,\ v\ :\ S \to T\ :\overset{\mu}{=}\ \{\sigma,\ \vec{c} \mapsto \delta'\}}\ \mathit{ConAss}$$

The third premise of this rule requires that δ' is typed by the translation of τ along v. Remember that we permit τ and δ to be $\bot$ and that we put $\bot^\mu = \bot$, i.e., if $\vec{c}$ is untyped, then δ' must be an untyped value. The last premise of the rule uses an auxiliary judgment to test whether the definition δ of $\vec{c}$ in S is compatible with the assignment δ'. $\gamma' \rhd_T \delta^v\ \leq\ \delta'$ holds immediately if $\delta = \bot$ because no incompatibility is possible, or if $\delta' = \top$ because any constant can be hidden. In all other cases, it holds iff $\gamma \rhd_T \delta^v \equiv \delta'$. (This rule inspired the notations $\bot$, $\top$, and $\leq$.)

Similarly, the following rule permits to add constants to theories. Note that it subsumes the cases of untyped and undefined constants.

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

The typing rules for morphisms are given in Fig. 7. They are the usual rules for identity and composition, and the variance of function types with respect to the inclusion of meta-theories.

4 Foundations

The typing rules of Mmt do not define the judgments for terms, which capture the semantics of a specific foundation. A foundation must define for $T \in \mathit{Thy}(\gamma)$ the judgments $\gamma \rhd_T \omega : \omega'$ (where we permit ω and ω' to be $\bot$) and $\gamma \rhd_T \omega \equiv \omega'$.

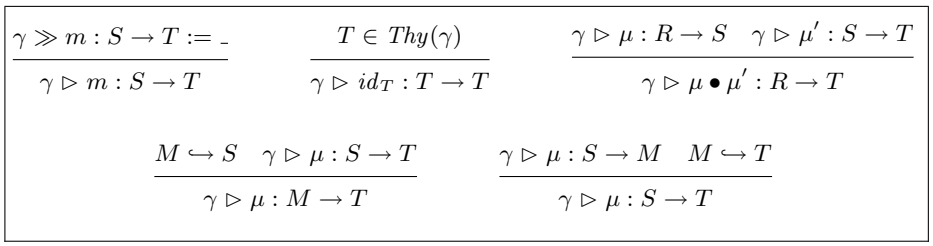


Fig. 7. Morphisms

A Foundation for OpenMath Fig. 8 first defines the judgment $\gamma; \Upsilon \rhd_T \omega$ for well-formed OpenMath expressions with variables from Υ. The rules $\mathcal{T}_{Var}$, $\mathcal{T}_{Con}$, $\mathcal{T}_\beta$, $\mathcal{T}_@$, and $\mathcal{T}_\alpha$ follow the OpenMath standard ([4]) treating Mmt theories as OpenMath content dictionaries. For $\mathcal{T}_\beta$, we have to interpolate the treatment of bound variables due to a gap the OpenMath standard: We permit every bound variable to occur in the attributions of every other variable bound by the same binder, which is useful for, e.g., mutually recursive let bindings.

Finally, $\mathcal{T}_\top$, $\mathcal{T}_\mu$, and $\mathcal{T}_\hookrightarrow$ formalize our extensions: $\top$, terms resulting from morphism applications (Υ^μ abbreviates component-wise morphism application.), and terms of the meta-theory are well-formed.

If $\gamma; \cdot \rhd_T \omega$, we say that ω is **structurally well-formed**. This judgment is used to define typing and equality of closed terms. Together with the rule *Con*, rule $\mathcal{T}_{type}$ expresses that all constants are untyped and that any well-formed term may occur as a definiens. Together with rule *ConAss*, rule $\mathcal{T}_{hide}$ permits the hiding of any constant. Rule $\mathcal{T}_{equal}$ expresses that two structurally well-formed terms are equal if their normal forms agree up to α-conversion.

The Maximal Foundation In the maximal foundation all typing and equality judgments between structurally well-formed terms, including all typing judgments involving $\bot$, hold. This does not represent a reasonable mathematical foundation, but it is useful because it subsumes any foundation that can possibly be useful. Since it is also very easy to implement, it can be used as a default foundation in practice when the actual foundation is not known or an implementation for it not available.

Definition 3. *A theory graph γ is **structurally well-formed** if it is clash-free and $\rhd \gamma$ holds relative to the maximal foundation.*

Structural well-formedness of γ guarantees that all references to modules, symbols, or variables exist and are in scope, and that all morphisms are well-typed. But it does not guarantee that all terms are well-typed. Thus, structural well-formedness provides an intermediate validation level between the very simple but rather weak validation against a context-free grammar – as done in web-oriented representation languages such as OMDoc and OpenMath – and the sophisticated but rather costly validation against a typing system. This is

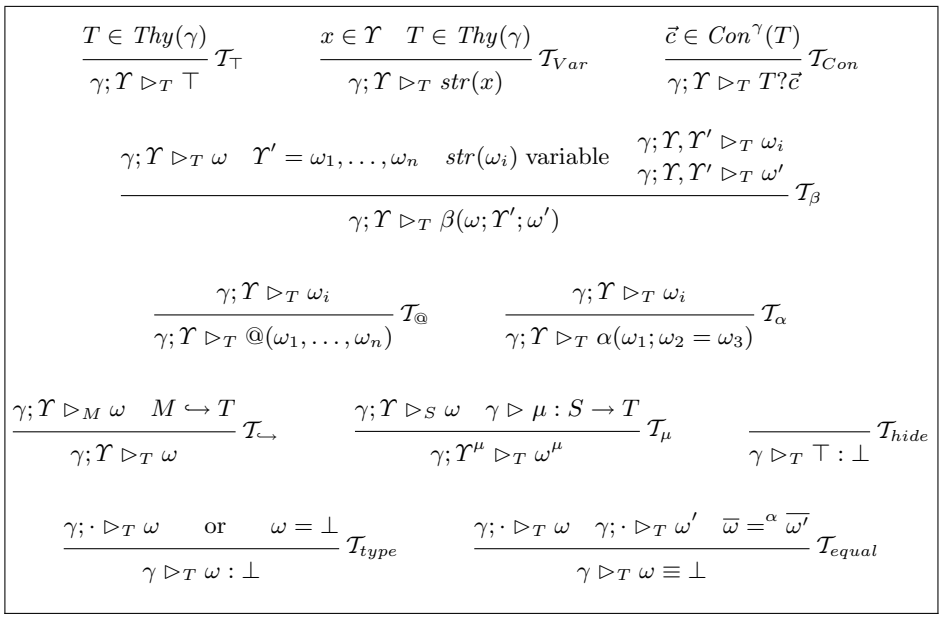


Fig. 8. Well-formed Terms

important for the development of MMT-based web services, for which structural well-formedness is typically a sufficient precondition.

Foundations for Type Theories For a given type theory TT, a foundation can be given in a straightforward way. First a theory M is declared containing declarations for the constants of TT itself. For example, $\texttt{STT} := \{\texttt{type}, \texttt{lambda}, \texttt{arrow}\}$ for simple type theory.

M should be published at a canonical URL (ideally equal to M) so that it can be used as a meta-theory. For example, $\texttt{HOL} \overset{\texttt{STT}}{:=} \{\texttt{o} : \texttt{STT?type}, \ldots\}$ could be the beginning of a theory for higher-order logic.

Then typing and equality are defined by giving the known rules of TT. For example, the typing judgment $\gamma \rhd_{\texttt{STT}} \bot : \texttt{STT?type}$ expresses that $\texttt{STT?type}$ is a well-formed MMT-type (Note that all terms, types, and kinds of TT are represented as MMT-terms, some of which may occur as MMT-types.), and the falsity of the judgment $\gamma \rhd_T \omega : \bot$ prevents untyped constants in T.

Then for any theory T, well-formedness can be determined by traversing the meta-theory relation upwards until a meta-theory is found for which a foundation is known. Thus, knowing the semantics of the theory M enables a human or a software system to understand any theory which has M as a (possibly indirect) meta-theory.

In our main example, from Sect. 2, a foundation for `LF` is enough to understand `monoid` as well as `integers`. Alternatively, a first-order automated prover

would use a foundation for `fol` (and ignore its meta-theory `LF`), and could be applied to problems over `monoid` but not to problems over `integers` – unless we give a view that translates from `integers` to a theory with meta-theory `fol` possibly hiding higher-order axioms used in `integers`.

5 Conclusion

We have introduced the MMT module system for mathematical theories. Its main design features are a simple, yet expressive language, foundation-independence, and scalability. That makes MMT a good candidate as an exchange language for logical knowledge.

The MMT grammar is quite simple capturing precisely the significant structuring primitives generally employed to form modules, symbols, and objects. This makes MMT expressive enough to be translated into, but at the same time simple enough to be translated out of relatively easily.

MMT is foundation-independent because the syntax of terms is uncommitted and the typing rules are parametric in the main judgments for terms. Using meta-theories and meta-morphisms, MMT permits to represent the foundation itself along with the used logic and the formalization carried out in that logic. Despite this lack of commitment, MMT can provide formal definitions of well-formedness and semantical equivalence.

Finally, MMT is scalable. Through the integration of URIs into the language, all modules and symbols (including induced ones) become addressable resources. This is used by elaboration to define the complete semantics of a (clash-free) theory graph in terms of functions that take URIs as arguments – and that can thus be realized as RESTful [8] HTTP GET to an MMT server. And the MMT type system decomposes theory graphs into simple units (i.e., symbols, assignments, modules without body) that can be communicated and manipulated sequentially, e.g., via streams, pipes, or HTTP POST requests. Communication can be carried out using an XML syntax for MMT (defined in [14]) that will become part of the OMDOC 2 [11] document format.

Furthermore, our treatment of MMT generalizes naturally to a management of change-oriented architecture while retaining a simple, fully formal typing system: By adding typing rules for deletions and updates, it becomes possible to reason about changes, and thus for software components to communicate only the difference between two theory graphs. If deletions and updates are realized as HTTP DELETE and PUT requests, respectively, we obtain a fully formalized RESTful web framework for mathematical documents.

It is non-trivial to add such a scalable interface to formal systems a posteriori. For example, the use of URIs fails if canonical names are not available for all induced symbols. Also most implementations of formal systems require the full knowledge base in main memory. In general, complex services for formal systems can typically only be realized with significant expertise and by adapting the reference implementation. For MMT, such applications are significantly simpler to realize and have in fact been implemented by us.

Acknowledgements MMT was designed in collaboration with Michael Kohlhase. A previous version of this paper appeared as [15].

References

1. S. Autexier, D. Hutter, H. Mantel, and A. Schairer. Towards an Evolutionary Formal Software-Development Using CASL. In D. Bert, C. Choppy, and P. Mosses, editors, *WADT*, volume 1827 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 1999.
2. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force, 2005.
3. Y. Bertot and P. Castéran. *Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.
4. S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See http://www.openmath.org/standard/om20.
5. CoFI (The Common Framework Initiative). *Casl Reference Manual*, volume 2900 (IFIP Series) of *LNCS*. Springer, 2004.
6. R. Constable, S. Allen, H. Bromley, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, 1986.
7. H. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.
8. R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
9. J. Goguen, Timothy Winkler, J. Meseguer, K. Futatsugi, and J. Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Applications of Algebraic Specification using OBJ*. Cambridge, 1993.
10. 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.
11. M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.
12. S. Owre, J. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992.
13. L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.
14. F. Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University Bremen, 2008.
15. F. Rabe and M. Kohlhase. An exchange format for modular knowledge. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, 2008.
16. D. Sannella and M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. In M. Karpinski, editor, *Fundamentals of Computation Theoryational*, pages 413–427. Springer, 1983.

A Type System

For simplicity, we assume that all theory graphs are clash-free so that some straightforward but tedious assumptions regarding names can be omitted from the rules. Parts in square brackets are optional.

Then the rules in Fig. 9 define theory graphs as list of modules. The rules Thy and $View$ add modules with empty body that are filled subsequently. We omit the analogue of rule $View$ for the case where S has no meta-theory. $ViewDef$ adds a view defined by a morphism.

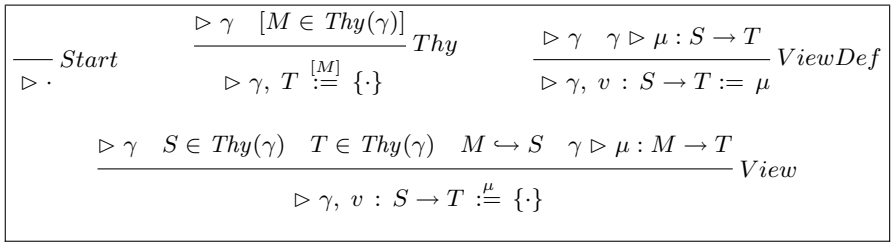


Fig. 9. Adding Modules

The rules in Fig. 10 add symbols to theories. The rule Con says that constant declarations $c : \tau := \delta$ can be added if δ has type τ. To avoid case distinctions, we use $\bot$ to indicate that τ or δ are omitted and assume that the foundation defines the typing judgment for such cases as well. The rules Str and $StrDef$ add an empty structure or a structure that is defined by a morphism in the same way as the rules $View$ and $ViewDef$. As for views, the case of rule Str where S has no meta-theory is omitted.

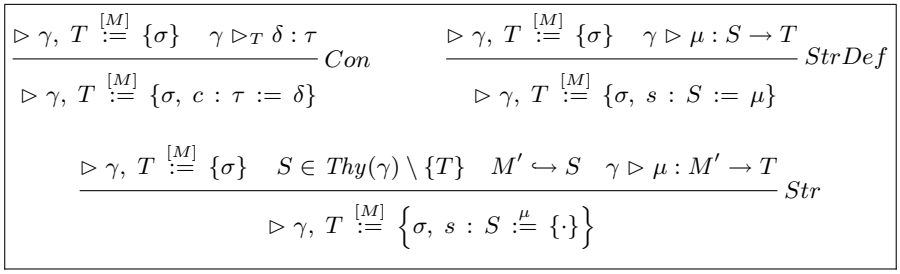


Fig. 10. Adding Symbols

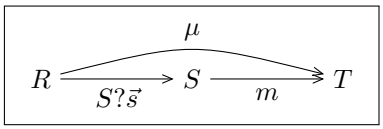
The addition of assignments to a link m is slightly more complicated because we unify the cases of adding an assignment to a structure or to a view. Let $last(\gamma, m : S \to T)$ denote that either

- $m = v$ and $\gamma = \ldots,\ v : S \to T \overset{[M]}{:=} \{\sigma\}$ or
- $m = T?s$ and $\gamma = \ldots,\ T \overset{[M]}{:=} \left\{\ldots,\ s : S \overset{[\mu]}{:=} \{\sigma\}\right\}$,

and in that case let $\gamma + Ass$ be the theory graph arising from γ by replacing σ with σ, Ass.

Then the rules in Fig. 11 add assignments. $ConAss$ adds an assignment $\vec{c} \mapsto \delta'$ for a constant $\vec{c}$ of S. Such assignments are well-typed if δ' is typed by the translation of the type of $\vec{c}$ along m. Again we avoid case distinctions by permitting $\tau = \bot$ and putting $\bot^m := \bot$. Furthermore, the value δ' must be compatible with the existing definition of $\vec{c}$ in the sense of rule $\leq ConAss$: If there is no such definition ($\delta = \bot$), this holds vacuously; if $\vec{c}$ is to be hidden ($\delta' = \top$), δ is irrelevant; in all other cases, δ translated along m must be equal to δ'.

The rule $StrAss$ is very similar to $ConAss$. The intended semantics of an assignment $\vec{s} \mapsto \mu$ for a structure $\vec{s}$ of S is that the diagram on the right commutes. This is the case if μ is well-typed as a morphism, and if μ is compatible with the existing constraints on $\vec{s}$ in the sense of rule $\leq StrAss$. This rule checks that any possibly existing definition for a symbol $\vec{c}$ of S is compatible with the induced assignment $\vec{s}/\vec{c} \mapsto (R?\vec{c})^\mu$.



Rule $\leq StrAss$ is in fact inefficient because all induced constants of R are checked. By exploiting the invariants of the module system, a more efficient version can be obtained (see [14]).

Finally Fig. 12 gives the typing rules for morphisms. The rule $\mathcal{M}_m$ handles links. $\mathcal{M}_{id}$ and $\mathcal{M}_\bullet$ give identity and composition of morphisms. Composition is written in diagrammatic order, i.e., from the domain to the codomain. Finally, $\mathcal{M}_{co}$ and $\mathcal{M}_{contra}$ are the usual co- and contravariance rules for functions.

$$\frac{\rhd \gamma \quad last(\gamma, m : S \to T) \quad \gamma \gg_S \vec{c} : \tau := \delta \quad \gamma \rhd_T \delta' : \tau^m \quad \gamma \rhd_T \delta^m \ \leq \ \delta'}{\rhd \gamma + \vec{c} \mapsto \delta'} \, ConAss$$

$$\frac{\delta = \bot \qquad \text{or} \qquad \delta' = \top \qquad \text{or} \qquad \gamma \rhd_T \delta^m \equiv \delta'}{\gamma \rhd_T \delta^m \ \leq \ \delta'} \leq ConAss$$

$$\frac{\rhd \gamma \quad last(\gamma, m : S \to T) \quad \gamma \gg S?\vec{s} : R \to S := \_ \quad \gamma \rhd \mu : R \to T \quad \gamma \rhd S?\vec{s} \ \leq^m \ \mu : R \to T}{\rhd \gamma + \vec{s} \mapsto \mu} \, StrAss$$

$$\frac{\gamma \rhd_T \delta^m \ \leq \ (R?\vec{c})^\mu \text{ whenever } \gamma \gg_S \vec{s}/\vec{c} : \_ := \delta}{\gamma \rhd S?s \ \leq^m \ \mu : R \to T} \leq StrAss$$

Fig. 11. Adding Assignments

$$\frac{\gamma \gg m : S \to T := \_}{\gamma \rhd m : S \to T} \, \mathcal{M}_m$$

$$\frac{T \in Thy(\gamma)}{\gamma \rhd id_T : T \to T} \, \mathcal{M}_{id} \qquad \frac{\gamma \rhd \mu : R \to S \quad \gamma \rhd \mu' : S \to T}{\gamma \rhd \mu \bullet \mu' : R \to T} \, \mathcal{M}_\bullet$$

$$\frac{M \hookrightarrow S \quad \gamma \rhd \mu : S \to T}{\gamma \rhd \mu : M \to T} \, \mathcal{M}_{contra} \qquad \frac{\gamma \rhd \mu : S \to M \quad M \hookrightarrow T}{\gamma \rhd \mu : S \to T} \, \mathcal{M}_{co}$$

Fig. 12. Morphisms