Structuring Theories with Implicit Morphisms

Florian Rabe[1,2] and Dennis Müller[2]

[1] LRI Paris
[2] FAU Erlangen-Nuremberg

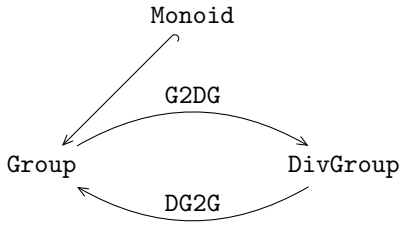
Abstract. We introduce *implicit* morphisms as a concept in formal systems based on theories and theory morphisms. The idea is that there may be at most one implicit morphism from a theory S to a theory T, and if S-expressions are used in T their semantics is obtained by automatically inserting the implicit morphism. The practical appeal of implicit morphisms is that they hit the sweet-spot of being extremely simple to understand and implement while significantly helping with structuring large collections of theories.

1 Introduction

Motivation Theory morphisms have proved an essential tool for managing collections of theories in logics and related formal systems. They can be used to structure theories and build large theories modularly from small components or to relate different theories to each other [**?**,**?**,**?**]. Areas in which tools based on theories and theory morphisms have been developed include specification [**?**,**?**], rewriting [**?**], theorem proving [**?**], and knowledge representation [**?**].

These systems usually use a logic L for the fine-granular formalization of domain knowledge, and a diagram D in the category of L-theories and L-morphisms for the high-level structure of large bodies of knowledge. This diagram is generated by all theories/morphisms defined, induced, or referenced in a user's development.

For example, a user might reference an existing theory `Monoid`, define a new theory `Group` that extends `Monoid`, define a theory `DivGroup` (providing an alternative formulation of groups based on the division operation), and then define two theory morphisms `G2DG` : `Group` ↔ `DivGroup` : `DG2G` that witness an isomorphism between these theories. This would result in the diagram on the right. Note that we use the syntactic direction for the arrows, e.g., an arrow $m : S \to T$ states that any S-expression E (e.g., a sort, term, formula, or proof) can be translated to an T-expression $m(E)$. Crucially, $m(-)$ preserves typing and provability.



The key idea behind implicit morphisms is very simple: We maintain an additional diagram I, which is commutative subdiagram of D and whose morphisms we call *implicit*. The condition of commutativity guarantees that I has at most

one morphism i from theory S to theory T, in which case we write $S \overset{i}{\hookrightarrow} T$. Commutativity makes the following language extension well-defined: if $S \overset{i}{\hookrightarrow} T$, then any identifier c that is visible to S may also be used in T-expressions; and if c is used in a T-expression, the semantics of c is $i(c)$ where i is the uniquely determined implicit morphism $i : S \to T$.

Despite their simplicity, the practical implications of implicit morphism are huge. For example, in the diagram above, we may choose to label `G2DG` implicit. Immediately, every abbreviation or theorem that we have formulated in the theory `Group` becomes available for use in `DivGroup` without any syntactic overhead. We can even label `DG2G` implicit as well if we prove the isomorphism property to ensure that I remains commutative, thus capturing the mathematical intuition that `Group` and `DivGroup` are just different formalizations of the same concept. While these morphisms must be labeled manually, any inclusion morphism like the one from `Monoid` to `Group` is implicit automatically.

Contribution We present a formal system for developing structured theories with implicit morphisms. Our starting point is the MMT language [**?**], which already provides a very general setting for defining and working with theories and morphisms. MMT is foundation-independent in the sense that it allows embedding a large variety of declarative languages (logics, type-theories, etc.). Therefore, all our results are foundation-independent — users of a particular language L can use MMT directly as a structuring formalism for L or can easily transfer our results to a dedicated implementation of L.

More precisely, out contribution is twofold.

Firstly, we give a novel presentation of the syntax and semantics MMT. Contrary to the original presentation in [**?**], which hard-coded a single structuring mechanism — there called *structures* — our definitions allows for adding arbitrary independent structuring mechanisms. This required a from-scratch redesign of the original presentation in [**?**] (which is why this paper has no preliminaries section).

Based on this extensible definition, we can introduce several additional structuring mechanisms. Many of these have already been implemented in the MMT tool but were not formally defined a formal definition of their semantics as a part of the MMT language had previously proved too complicated to spell out elegantly. With our reformulation, this is now not only possible but very easy and elegant. As an example, we give includes and unions of theories, which induce the most important special cases of implicit morphisms.

Secondly, we describe implicit morphisms as a novel feature of MMT and give a number of examples for obtaining and using implicit morphisms. In particular, we recover the concept of realms [**?**] as a special case of implicit morphisms.

In fact, implicit morphisms work so well that we have refactored the MMT tool in such a way that implicit morphisms are now more primitive than inclusion morphisms. The semantics of inclusion morphisms is obtained by saying that inclusions are implicit morphisms that map all identifiers to themselves. Even the fundamental property that a theory may reference its own identifiers is then

just a consequence of the fact that all identity morphisms are implicit. Therefore, surprisingly, adding the new feature of implicit morphisms to the MMT kernel has made its design much simpler.

2 Theories and Theory Morphism

[1] EdN:1

2.1 Overview

Modularity in MMT As much as possible, theories and morphisms are treated uniformly, and we use the word **module** to refer to them collectively. MMT allows constructing large modules from small ones in two ways:

- **structured modules** contain declarations that import other, previously defined ones,
- **module expressions** are anonymous expressions (akin to formulas, terms, etc.) that denote modules.

These two kinds of constructions have very different advantages but the same expressivity. For example, we can construct the theory of commutative groups as the structured theory $\texttt{CommutativeGroup} = \{\textbf{include}\ Group, \textbf{include}\ CommutativeMagma\}$ or as the theory expression $\texttt{Group} \cup \texttt{CommutativeMagma}$. The former is more practical when building named toplevel theories in software systems because each include is a separate declaration that can be parsed, checked, and flattened individually. While negligible in this example, this becomes relevant quickly in large theories with complex structure. The latter is more practical when building anonymous theory that are only needed temporarily. In software systems, they also allow reusing operations like equality and λ-abstraction that already be implemented at the level of the base logic. Building large morphism is as important as building large theories: for example, we want to be able to build morphisms out of `CommutativeGroup` by combining compatible morphisms out of `Group` and `CommutativeMagma`.

Flat Modules MMT provides formal syntax for both structured modules and module expressions and defines their semantics via **flattening**, which defines for every module M a flat module $M^\flat$.

Flat theories are lists of declarations $c[: E][= e]$ where E and e are optional expressions. We write $\texttt{dom}(T)$ for the set of constant identifiers c in $T^\flat$ and $\texttt{Obj}(T)$ for the set of closed expressions using only the symbols $c \in \texttt{dom}(T)$. Constant declarations subsume virtually all basic declarations common in formal

[1] EdNote: @Dennis: can you formally work through the example from the intro in here? That would be several example environments for (i) after introducing the syntax of theories, the theory Group and DivGroup (one in detail, one sketched), (ii) after introducing morphisms the two isomorphisms (one in detail, one sketched), (iii) after introducing includes, the refactoring of Group using an include, (iv) examples for how the refactored Group is flattened

systems such as type/function/predicate symbols, axioms, theorems, inference rules, etc. In particular, theorems can be represented via the propositions-as-types correspondence as declarations $c : F = P$, which establish theorem F via proof P. Similarly, MMT expressions subsume virtually all objects common in formal systems such as terms, types, formulas, proofs.

Individual formal languages arise as fragments of MMT: they single out the well-formed expressions by defining the two MMT-**judgments** $\vdash_T e : e'$ (typing) and $\vdash_T e \equiv e'$ (equality) for every theory T and $e, e' \in \texttt{Obj}(T)$. The details can be found in [**?**].

Flat morphisms from a theory S to a theory T are lists of assignments $c := e$ where $c \in \texttt{dom}(S)$ and $e \in \texttt{Obj}(T)$. Every morphism M induces a **homomorphic extension** $M(-) : \texttt{Obj}(S) \to \texttt{Obj}(T)$, which replaces every $c \in \texttt{dom}(S)$ in an S-expression with the T-expression e such that $c := e$ in M.

The declarations in theories and morphisms are subject to typing conditions that are not essential for our purposes here. We only mention the main theorem that guarantees that well-typed morphisms preserve judgments, i.e., if $\vdash_S e : E$, then $\vdash_T M(e) : M(E)$. This includes the preservation of truth via the propositions-as-types principle where E is a proposition and e its proof.

An MMT **diagram** consists of a set of named structured module declarations and induces sets of anonymous module expressions. These are mutually recursive: the names of the former provide the base cases for inductively forming the latter, and the latter may occur in the bodies of the former. Every diagram induces sets
- $\texttt{thy}$ of identifiers of theories t,
- $\texttt{Thy} \supseteq \texttt{thy}$ of theory expressions T,
- $\texttt{mor}(S, T)$ of identifiers of atomic morphisms m from S to T, and
- $\texttt{Mor}(S, T) \supseteq \texttt{mor}(S, T)$ of morphism expressions M from S to T.

Then flattening assigns
- to each $T \in \texttt{Thy}$ the flat theory $T^\flat$,
- to each $M \in \texttt{Mor}(S, T)$ the morphism $M^\flat$ from $S^\flat$ to $T^\flat$.

Well-Formedness The inference system to define well-formed modules, and expressions is not a primary interest of this paper. Therefore, we only mention those aspects that are relevant later on.

However, one detail will be critical later on: the inference rules for the well-formedness of expressions include the following base case for constants:

$$\frac{c : E[= e] \text{ in } T^\flat}{\vdash_T c : E} \qquad \frac{c[: E] = e \text{ in } T^\flat}{\vdash_T c = e}$$

Correspondingly, the definition of the homomorphic extension of a morphism includes the following base case for constants:

$$M(c) = e \text{ where } c := e \in M^\flat$$

These base cases introduce a mutual recursion between well-formedness and flattening: the well-formedness of a declaration in a theory depends on the flattening of all preceding declarations; and the well-formedness of an assignment in

a morphism depends on the homomorphic extension of the morphism obtained by flattening of all preceding assignments. Vice versa, well-formedness is a precondition for defining the flattening. It is desirable to define well-formedness independently of flattening. But the mutual recursion makes sense from an implementation perspective: typical tools for formal systems will first parse, check, and flatten the first declaration, then continue with the next declaration. In fact, often the checking of well-formedness is as expensive as flattening anyway.

Therefore, we will make flattening a partial function, i.e., $X^\flat$ is defined iff the module X is well-formed.

2.2 Syntax

We start with the syntax for theories (which arises as a special case of the one given in [**?**]):

Definition 1 (Theory). *A **theory** T is of the form*

$TDec$	::=	$t = \{Dec, \ldots, Dec\}$	*theory declaration*
Dec	::=	$n[: o][= o]$	*constant declaration*
c	::=	$t?n$	*qualified constant identifiers*
o	::=	$c \mid \ldots$	*expressions built from constants*
T	::=	t	*theory expressions*

*In a theory declaration, each symbol **name** n may be declared only once, and its **type** and **definiens** (if present) must be closed expressions only previously introduced constants. We omit the remaining productions for expressions here, which allow forming complex expressions using application, binding, variables, literals, etc.*

Example 1. The (flat) theory for `Group` and `DivGroup` as presented in the introduction are:

```
theory Group : ?Meta =
  universe      : type          | # U
  op            : U → U → U     | # 1 ∘ 2
  unit          : U             | # e
  inverse : U → U | # 1 ^-1

  axiom_assoc         : ⊢ ∀[x]∀[y]∀[z] x ∘ (y ∘ z) ≐(x ∘ y) ∘ z
  axiom_right_unit    : ⊢ ∀[x] x ∘ e ≐x
  axiom_left_unit     : ⊢ ∀[x] e ∘ x ≐x
  axiom_inverse : ⊢ ∀[x] x ∘ (x^-1) ≐e

theory DivisionGroup : ?Meta =
  universe : type | # U prec 1
  div : U → U → U |# 1 / 2
  nonEmpty : U | # e prec 1
  // axioms ...
```

where ▮, |, ▮ are delimiters and the # <not> components are (optional) notations. ?Meta is an intended meta-theory containing e.g. the symbols $\forall, \doteq$ etc.

At this point, this grammar only allows forming flat theories, i.e., `Thy` = `thy`. It is intentionally independent of the concrete choice of modularity principles, i.e., the kinds of module-structuring declarations (e.g., includes) and the module-expression-forming operators (e.g., union). But it already introduces all non-terminals for modularity: new declaration kinds for structured theories arise by adding production for the non-terminal *Dec* and new theory expressions are formed by adding productions for the non-terminal T. This makes it easy to incrementally add new declaration kinds and operators to the language.

We proceed accordingly for flat morphisms:

Definition 2 (Morphism). *A **morphism** is of the form*

$MDec$	$::=$	$m : T \to T = \{Ass, \ldots, Ass\}$	*flat morphism declaration*
Ass	$::=$	$c := o$	*assignment to symbol*
M	$::=$	m	*morphism expressions*

such that a morphism declaration contains exactly one assignment $c := o$ for each $c \in \texttt{dom}(S)$, all of which satisfying $o \in \texttt{Obj}(T)$.

Example 2. We can now specify the morphisms between our two definitions of groups. The theory of groups as monoids with inverses is given in Example 3, the theory `DivGroup` and the morphism `DG2G` are as follows:

```
view DG2G : ?DivisionGroup -> ?Group =
  universe = universe ▮
  div = [a,b] a ∘ (b^-1) ▮
  nonEmpty = e ▮
  // axioms ... ▮
▮
```

The morphism `DG2G` hence maps the universe of a division group to the universe of a group, division to the expression $\lambda a, b.\ a \circ b^{-1}$ and the witness `nonEmpty` of a division group to the unit e of a group. Additionally, the view has to map the axioms of a division group to their *proofs* in `Group` of their translated statements.

The reverse morphism `G2DG` is completely analogous, mapping `op`, `unit` and `inverse` to their corresponding definitions in terms of the division operation `div`.

Finally, we define diagrams as collections of modules:

Definition 3 (Diagram). *A **diagram** is a list of theory and morphism declarations:*

$$Dia \quad ::= \quad (TDec \mid MDec)^* \qquad \textit{diagrams}$$

A diagram is well-formed if each atomic theory/morphism declaration has a unique name is well-formed relative to the diagram preceding it.

We give some examples how to add modularity principles to the grammar:

Example 3 (Includes). We extend the grammar with

Dec	::=	**include** S	include a theory into a theory
Ass	::=	**include** M	include a morphism into a morphism

This allows writing the theory `Group` as in Example 1 by extending a theory of monoids:

```
theory Monoid : ?Meta =
  universe        : type          | # U
  op              : U → U → U     | # 1 ∘ 2
  unit            : U             | # e

  axiom_assoc          : ⊢ ∀[x]∀[y]∀[z] x ∘ (y ∘ z) ≐(x ∘ y) ∘ z
  axiom_right_unit     : ⊢ ∀[x] x ∘ e ≐x
  axiom_left_unit      : ⊢ ∀[x] e ∘ x ≐x

theory Group : ?Meta =
  include ?Monoid

  inverse : U → U | # 1 ⁻¹
  axiom_inverse : ⊢ ∀[x] x ∘ (x⁻¹) ≐e
```

The flat variant $\texttt{Group}^\flat$ we get by making all included symbols (from `Monoid`) available to `Group`, resulting in exactly the theory as presented earlier in Example 1.

Example 4 (Union). We extend the grammar with

T	::=	$T \cup T$	union of theories
M	::=	$M \cup M$	union of morphisms

Example 5 (Category of Theories). To obtain the category structure of theories and morphisms, we need identity and composition. We add them to the grammar as follows:

M	::=	id_T	identity morphism
M	::=	$M;M$	morphism composition

For each example, we will define the semantics in the next section.

2.3 Semantics

Flattening Just like the syntax, the definition of flattening is **compositional** in the sense that new modularity principles can be added later independently of each other. In particular, for every module expression X, we define its flattening $X^\flat$ by induction on the syntax. This follows the principle of compositional semantics, e.g., $(f(X,Y))^\flat$ depends only on $X^\flat$ and $fltY$.

For theories T, at this point, this induction only has the base case of a reference to a declared theory t. Further cases arise when we add structuring declarations.

For the base case, of a theory $t = \{\Sigma\}$, we define $t^\flat$ by induction on the declarations in Σ:

$$t^\flat = \Sigma^\flat$$

where

$$\cdot^\flat = \varnothing$$

$$(\Sigma, D)^\flat = \Sigma^\flat \cup (\Sigma^\flat * D)^\flat$$

Here $(T * D)^\flat$ is the flattening of declaration d in the context of theory T.

At this point, we only have one case for declarations D, namely constant declarations. Their flattening is trivial:

$$(\Sigma * n[: E][= e])^\flat = \{t?n[: E][= e]\}$$

where t is the name of the containing theory.

Correspondingly, for a declared morphism $m : S \to T = \{\sigma\}$, we have $m \in \texttt{Mor}(S, T)$ and $m^\flat$ is defined by induction on the assignments in σ:

$$\cdot^\flat = \varnothing$$

$$(\sigma,\ A)^\flat = \sigma^\flat \cup (\sigma^\flat * A)^\flat$$

with the trivial base case

$$(\sigma * c := e)^\flat = \{c := o\}$$

Example 6 (Includes (continued from Ex. 3)). Includes add new declarations D, so we have to add cases to the definition of $(\Sigma * D)^\flat$. We do this as follows

$$(\Sigma * \textbf{include}\ S)^\flat = S^\flat$$

This has the effect of copying over all declarations from the included into the including theory.

Note that the definition of $(\Sigma * n : E = e)^\flat$ qualifies the constant name n with the theory name t to form the identifier $t?n$ in the flattening. Therefore, includes can never lead to name clashes.

Also note, that because $S^\flat$ is a *set* of declarations, the include relation is transitive: if t includes s via two different paths, $t^\flat$ only contains one copy of the declarations of $s^\flat$. The situation is slightly more complicated for morphisms: if a morphism out of t includes two different morphisms out of s, these have to agree. Therefore, flattening and well-formedness must be distinguished:

$$(\sigma * \textbf{include}\ M)^\flat = M^\flat$$

under the condition $M^\flat$ agrees with $\sigma^\flat$ on any constant that is in the domain of both.

Example 7 (Union (continued from Ex. 4)). We define the semantics of unions of theories as

$$(S \cup T)^\flat = S^\flat \cup T^\flat$$

The handling of name clashes is the same as for includes: Because the theories declared in a diagram have unique names and declarations in the flattening are qualified by theory names, union theories cannot have name clashes. But we have to watch out when taking unions of morphisms: We define $M_1 \cup M_2 \in \mathtt{Mor}(S_1 \cup S_2, T)$ if $M_i \in \mathtt{Mor}(S_i, T)$ and M_1 and M_2 agree for every constant in $\mathtt{dom}(S_1^\flat) \cap \mathtt{dom}(S_2^\flat)$. In that case, we put

$$(M \cup N)^\flat = M^\flat \cup N^\flat$$

Finally, we define the semantics of the category structure:

Example 8 (Category of Theories (continued from Ex. 5)). For $T \in \mathtt{Thy}$, we put $id_T \in \mathtt{Mor}(T, T)$ and define

$$id_T^\flat = \{c := c \mid c \in \mathtt{dom}(T^\flat)\}$$

For $R, S, T \in \mathtt{Thy}$ and $M \in \mathtt{Mor}(R, S)$ and $N \in \mathtt{Mor}(S, T)$, we put $M; N \in \mathtt{Mor}(R, T)$ and define

$$M; N^\flat = \{c := N^\flat(M^\flat(c)) \mid c \in \mathtt{dom}(R^\flat)\}$$

2.4 Semantic Properties

For two theories $S, T \in \mathtt{Thy}$, we say that S is **included** into T, written $S \hookrightarrow T$, if $S^\flat \subseteq T^\flat$. Thus, every declaration of S is also available in T and $\mathtt{Obj}(S) \subseteq \mathtt{Obj}(T)$. This induces an inclusion morphism, which maps every $c \in \mathtt{dom}(S)$ to itself. We do not introduce a name for that morphism and simply define $id_S \in \mathtt{Mor}(S, T)$ whenever $S \hookrightarrow T$ (which includes the special case $S = T$). Moreover, if $S \hookrightarrow T$ and $M \in \mathtt{Mor}(T, U)$, we define the restriction $M|_S \in \mathtt{Mor}(S, U)$ of M to S by $id_S; M$.

We say that two theories $S, T \in \mathtt{Thy}$ are **equal**, written $S \equiv T$, if $S^\flat = T^\flat$. This is equivalent to $S \hookrightarrow T$ and $T \hookrightarrow S$.

Correspondingly, we say that two morphisms $M, N \in \mathtt{Mor}(S, T)$ are **equal**, written $M \equiv N$, if $M^\flat = N^\flat$. In that case, $M(e) = N(e)$ for all expressions $e \in \mathtt{Obj}(S)$.

Clearly, $\equiv$ is an equivalence relation and $\hookrightarrow$ is an order relation (if anti-symmetry is taken with respect to $\equiv$).

Example 9 (Includes (continued from Ex. 6)). We have the following soundness theorem for includes: if a theory is declared as $t = \{\Sigma\}$ and Σ contains **include** S, then $S \hookrightarrow t$.

Note that in the absence of include declarations or unions, $s \hookrightarrow t$ never holds because all $c \in \mathtt{dom}(s)$ are of the form $s?n$ and all $c \in \mathtt{dom}(t)$ are of the form $t?n$. In fact, $s^\flat$ and $t^\flat$ unless $s = t$.

Example 10 (Union (continued from Ex. 7)). We have the following soundness theorem for unions: $S_i \hookrightarrow S_1 \cup S_2$. Moreover, $S_1 \cup S_2$ is a colimit in the category of theories and morphism, and $M_1 \cup M_2 \in \texttt{Mor}(S_1 \cup S_2, T)$ is the universal morphism out of it.

Example 11 (Category of Theories (continued from Ex. 8)). Identity and composition are congruent with respect to $\equiv$. Moreover, composition is associative and identity neutral with respect to $\equiv$. Thus, each diagram indeed yields the structure of a category with objects `Thy` and arrows $\texttt{Mor}(S, T)$.

3 Implicit Morphisms

3.1 Overview

The motivation behind implicit morphisms is to generalize the benefits of include declarations even further. Sometimes an atomic morphism is conceptually similar to an inclusion: For example, we might want to declare $m : \texttt{Division} \to \texttt{Group} = \{\texttt{divide} := \lambda x, y.x \circ y^{-1}, \ldots\}$ to show how division is "included" into the theory of groups. Other times we have to use structures because we need to duplicate declarations but we would still like to use them like includes. For example, we have to declare the theory of rings using two structures (for the commutative group and the monoid). But we still want a ring to be implicitly converted into, e.g., a commutative group.

Our key idea is to use a commutative[3] subdiagram of the MMT diagram, which we call the *implicit-diagram*. It contains all theories but only some of the morphisms – the ones designated as *implicit*. Because the implicit-diagram commutes, there can be at most one implicit morphism from S to T – if this morphism is M, we write $S \overset{M}{\hookrightarrow} T$.

The implicit-diagram generalizes the inclusion relation $\hookrightarrow$. In particular, all identity and inclusion morphisms are implicit, and we recover $S \hookrightarrow T$ as the special case $S \overset{id_S}{\hookrightarrow} T$. Moreover, the relation "exists M such that $S \overset{M}{\hookrightarrow} T$" is also an order.

Consequently, many of the advantages of inclusions carry over to implicit morphisms.

- It is very easy to maintain the implicit-diagram, e.g., as a partial map that assigns to a pair of theories the implicit morphism between them (if any).
- We can generalize the visibility of identifiers: If $S \overset{M}{\hookrightarrow} T$, we can use all S-identifiers in T as if S were included into T. Any $c \in \texttt{dom}(S)$ is treated as a valid T-identifiers with definiens $M(c)$.
- We can still use canonical identifiers $s?n$. Because there can be at most one implicit morphism $s \overset{M}{\hookrightarrow} T$, using $s?n$ as an identifier in T is unambiguous. Crucially, we can use $s?n$ without bothering what M is.

[3] A diagram commutes if for any two morphisms $M, M' \in \texttt{Mor}(S, T)$, we have $M \equiv M'$.

3.2 Syntax

We introduce the family of sets $\texttt{Mor}^{\texttt{i}}(S,T)$ as a subset of $\texttt{Mor}(S,T)$, holding the *implicit* morphisms. The intuition is that $\texttt{Thy}$ and $\texttt{Mor}^{\texttt{i}}(S,T)$ (up to equality of morphisms) form a thin broad subcategory of $\texttt{Thy}$ and $\texttt{Mor}(S,T)$.

It remains to define which morphisms are implicit. For that purpose, we allow MMT declarations to carry *attributes*. Precisely, we add the following productions

$MDec$	::=	$Att\ MDec$	attributed morphism
Dec	::=	$Att\ Dec$	attributed declaration
Att	::=	**implicit** $\mid \ldots$	attributes

The set of attributes is itself extensible, and the above grammar only list the one that we use to get started. Additional attributes may be added with the condition that any additional attribute increases the set of implicit morphisms.

Specifically, the set $\texttt{Mor}^{\texttt{i}}(S,T) \subseteq \texttt{Mor}(S,T)$ of **implicit** morphisms contains the following elements:

- all identity morphisms id_T,
- all compositions $M;N$ of implicit morphisms M and N,
- all declared morphisms $m : S \to T = \{\sigma\}$ whose declaration carries the attribute **implicit**.

3.3 Semantics

We only have to make two minor changes to the semantics to accommodate implicit morphisms. The first change govern how we obtain implicit morphisms, the second one how we use them.

Obtaining We define well-formedness for our extended syntax. We define that an attribute to a declaration is only well-formed if the category of implicit morphisms remains thin, i.e., for any two theories, there may be only one (up to equality) implicit morphism between them. In other the diagram containing only the implicit morphisms must commute.

Depending on what modularity principles we add, equality of morphisms may or may not be decidable. Therefore, in the general case, implementations may have to employ sound but incomplete criteria for commutativity (see Sect. 3.4).

Using We make a small modification to the rules in

Definition 4 (Visible Declarations). *The set $T^{\flat i}$ contains the following declarations: for every $S \xhookrightarrow{M} T$ and every $c[:t][=d]$ in $S^\flat$, the declaration $c[:M(t)] = M(c)$.*

The set $\texttt{Obj}^i(T)$ contains all objects formed from the symbols in $T^{\flat i}$. The map $M(-)$ maps every c for which $M^\flat$ does not provide an assignment to $M(d)$ where d is the definiens of c.

This modification is conservative in the sense that $T^\flat \subseteq T^{\flat i}$ and $T^{\flat i} \setminus T^\flat$ contains only declarations with definiens. Moreover, $\texttt{Obj}(T) \subseteq \texttt{Obj}^i(T)$ and $M(-)$ remains unchanged on all $o \in \texttt{Obj}(T)$. It is easy to see that $M(-)$ still preserve all judgments.

3.4 Commutativity

To prove commutativity, we have to find all paths in the implicit-diagram and check $M \equiv M'$ for all pairs of morphisms between the same nodes.

MMT reduces $M \equiv M'$ for $M, M' \in \texttt{Mor}(S, T)$ to a first-order problem in the theory of categories, which uses the following axioms:

- associativity of composition,
- neutrality of identity,
- $M; M' \equiv id_S$ as well as $M'; M \equiv id_T$ if $M \in \texttt{Mor}(S, T)$ and M' is the known *inverse of M,
- if an atomic morphism m includes $M \in \texttt{Mor}(R, T)$, then $M|_R \equiv M$.

This is a sound but not complete axiomatization of $M \equiv M'$. We can obtain a complete axiomatization in a totally different way: Check $\vdash_T M(c) \equiv M'(c)$ for all $c \in \texttt{dom}(S)$. However, this can be extremely expensive: It requires computing $\texttt{dom}(S)$ and proving an object level equality for each identifier. Even if the equality of objects is decidable, this is often too expensive.

In fact, because implicit morphisms are meant to be used in very particular cases, we expect the above incomplete calculus to perform reasonably well in practice. If showing the equality of certain morphisms is hard, one should probably not make them implicit in the first place.

3.5 Inverse Morphisms

We only two use sufficient criteria to determine an *inverse of a morphism M.

Conservative Extensions Intuitively, t extends S definitionally, if it only adds defined symbols, e.g., abbreviations or theorems. That is the most important case of conservative extensions.

Definition 5 (Definitional Extension). *An include declaration*

$$t = \{\textbf{invertible include } S, \Sigma\}$$

*is **definitional** if*

- *all symbol declarations in Σ have a definiens,*
- *all include declarations* **include** *R in Σ are such that R also definitionally includes S.*

There are multiple, subtly different definitions of conservative extension. However, definitional extensions as defined above are always conservative (i.e., for any reasonable definition of conservativity and in any reasonable formal language). In fact, they are always invertible: The inverse morphism $i : t \to S$ maps

– an identifier $c \in \texttt{dom}(S)$ to c and everythemselves and the others to their definiens,
– any other identifier, which must have a definiens d, to $i(d)$.

Renamings Intuitively, an atomic morphism $m : S \to T$ is a renaming if it maps all S-identifiers to T-identifiers (rather than T-objects).

Definition 6 (Renaming). *Consider a morphism $M \in \texttt{Mor}(S,T)$. Let $A \subseteq \texttt{dom}(S)$ and $B \subseteq \texttt{dom}(T)$ contain those identifiers without definiens. M is an (injective, surjective) **renaming** if the restriction of $M(-)$ is an (injective, surjective) mapping $A \to B$.*

Clearly, if m is an injective and surjective renaming, it is straightforward to obtain its inverse by inverting $M(-)$.

4 Applications

Canonical isomorphisms.

User-declared atomic morphisms.

Extending Theories from the Outside

Transparent Refactoring A major drawback of OpenMath/MMT-style qualified identifiers is that it can preclude transparent refactoring. For example, consider a theory $t = \{\textbf{include}\ r, \textbf{include}\ s\}$ and assume we want to move a symbol n from r to s. This requires changing any qualified reference to $r?n$ (in any theory including t) to $s?n$. That can be very costly, especially in large libraries where the refactorer might not even know of or have write access to all theories importing t.

With implicit morphisms, we rename s to s', move n from s' to r, and recover s by $s = \{\textbf{include}\ s', n\}$. Now we can change define $t = \{\textbf{include}\ r, \textbf{include}\ s'\}$ and give an implicit morphism from s to t, which maps $s?n$ to $r?n$. All existing references to $s?n$ stay well-formed (and serve as aliases for $r?n$) so that no further changes in theories importing t are needed.

5 Related Work

5.1 Realms

[**?**] defines a realm (essentially) as a certain cluster of theories:
– a set of isomorphic theories $B_1, \ldots,$
– for each B_i, a set of retractable extensions $E^i_1, \ldots,$
– a theory F including all of the above.

The intuition is that the B_i are alternative definitions of a theory; the E^i_j add definitions and theorems; and F aggregates everything into the outwardly visible interface (the face) of the realm.

A good example is the realm of topological spaces with the B_i corresponding to the various possible definitions (via neighborhoods, open sets, closure operator, etc.).

Users of the realm include only the face and thus (i) gain access to all derived knowledge in the theory of topological spaces, and (ii) do not have to commit to one particular definition. Indeed, this corresponds more closely to the way how conventional mathematics sees the theory of topological spaces.

[**?**] calls for an implementation of realms as a new primitive concept in addition to theories and morphisms. However, in the presence of implicit morphisms, we do not need it: All we have to do is to designate the isomorphisms and the retraction is implicit and include any one of the B_i into F.

5.2 Canonical Structures in Coq

[**?**]

5.3 Implicit Conversions in Scala

The Scala programming language [**?**] has a clean implementation of implicit conversions between any two types. If we think of functions between classes as theory morphisms, this corresponds to implicit theory morphisms.

Contrary to MMT, a conversion is only available if explicitly imported into the current namespace. Moreover, Scala does not chain conversions, i.e, the composition of implicit morphisms is not implicit. Scala's limitations are intentional: Their goal is to prevent programmers from overusing implicit conversions because that can be very confusing to readers.

These limitations are lifted in MMT's conversions can only act between theories anyway. Thus, their potential to confuse is smaller. Moreover, the use of qualified identifiers in MMT means that readers can always tell where an identifier is implicitly-included from, and the MMT IDE can always show the implicit morphism along which it is imported.

EdN:2 [2]

6 Conclusion

References

[2] EDNOTE: drawback: global data structure