Morphism Equality in Theory Graphs

Florian Rabe[https://orcid.org/0000-0003-3040-3655] and Franziska Weber[https://orcid.org/0000-0003-3379-5922]

University Erlangen-Nuremberg, Germany

Abstract. Theory graphs have theories as nodes and theory morphisms as edges. They can be seen as generators of categories with the nodes as the objects and the paths as the morphisms. But in contrast to generated categories, theory graphs do not allow for an equational theory on the morphisms. That blocks formalizing important aspects of theory graphs such as isomorphisms between theories.
MMT is essentially a logic-independent language for theory graphs. It previously supported theories and morphisms, and we extend it with morphism equality as a third primitive. We show the importance of this feature in several elementary formalizations that critically require stating and proving certain non-trivial morphism equalities. The key difficulty of this approach is that important properties of theory graphs now become undecidable and require heuristic methods.

1 Introduction and Related Work

Theory Graphs and Motivation Logical **theories** are an essential meta-level concept for encapsulating a set of declarations and axioms. For example, the theory `Group` of first-order logic declares a base set U, a binary operation $\circ$ on U, a unary operation $^{-1}$, a neutral element e, and the usual axioms.

Theory morphisms extend this to a category. A theory **morphism** from S to T interprets S in T or, equivalently, constructs a model of S if provided a model of T. Formally, it maps every constant declared in S to a T-expression in a way that the homomorphic extension preserves all types and theorems. For example, we can define the theory `DivGroup` of division groups, an alternative way to define groups, using a binary division operation $/$. Then we can define a theory morphism $\texttt{GtoDG} : \texttt{Group} \longrightarrow \texttt{DivGroup}$ by mapping $x \circ y$ to $x/(e/y)$, x^{-1} to e/x and e to e and proving all `Group` axioms of the thus-defined group.

The category of theories has proved extremely valuable in the large scale structuring of mathematical formalizations, especially when combined with module systems [FGT92,SW83,AHMS99]. A diagram in this category is also called a **theory graph**. Many formal systems provide support for constructing large theory graphs over various logics, including proof assistants such as IMPS [FGT93] and PVS [ORS92], specification systems such as Hets [MML07] and Specware [SJ95], and logical frameworks such as LF [RS09] and Isabelle [KWP99].[1]

[1] The individual theory graphs are implicit in the respective libraries and usually not the subject of specific publications.

However, maybe surprisingly, none of these systems includes support for **morphism equality**. In category theory, a diagram consists of three components: a set of objects (nodes of the theory graph), a set of atomic morphisms between objects (edges), from which the set of morphisms (paths) is generated by composition, and a set of pairs of equal morphisms between the same objects (equality of two paths between the same nodes). But practical systems for theory graphs have restricted attention to the former two components even though the latter is critical for the diagram chase–style arguments that are a hallmark of category theory.

For example, we can give a second morphism $\texttt{DGtoG} : \texttt{DivGroup} \longrightarrow \texttt{Group}$ that maps (among others) x/y to $x \circ y^{-1}$. We can then see that $\texttt{DGtoG};\texttt{GtoDG} = id_{\texttt{DivGroup}}$ and $\texttt{GtoDG};\texttt{DGtoG} = id_{\texttt{Group}}$, i.e., that `Group` and `DivGroup` are isomorphic. If we encode theories and morphisms in some type theory (e.g., as record types and functions or as signatures and functors), it is often possible to encode these equalities correspondingly.

But existing systems that specifically work with theory graphs and the category induced by them, do not make it possible, let alone easy, to express, prove, and use such equalities as a part of the theory graph development. Our goal is to design a theory graph language that supports defining theories and morphisms and proving equalities of morphisms.

Application to Realms A parallel motivation of our work was provided by the goal of formalizing realms.

Working with multiple isomorphic formalizations of the same mathematical theory is often both unavoidable and cumbersome. In 2014, Carette, Farmer, and Kohlhase introduced the concept of *realms* [CFK14] as a high-level structuring feature for formal mathematics. Their basic idea is to provide an abstraction layer at which multiple isomorphic formalizations are identified. For example, users should be able to ignore the difference between `Group` and `DivGroup`: when creating a group, giving either $\circ$ or / should suffice; and when using a group, both should be available.

However, while the idea of realms was well-received (best paper award), neither [CFK14] nor any follow-up work conducted a detailed investigation of how realms should be implemented in a practical system.

In a work-in-progress paper [RW22], we partially formalized several examples that were given informally in [CFK14]. A key result of these case studies was that theory graph languages with morphism equality are needed to formalize realms. This motivated the present paper, in which we introduce such a language.

Contribution and Overview We start with the Mmt language [RK13], which already allows defining theories and morphisms. Mmt is independent of the base language, but some assumptions about the base language are needed to state equalities — therefore, we work with Mmt's instantiation with the logical framework `LF` [HHP93].

Our main contribution is introducing, in Sect. 4, morphism equalities to Mmt. Concretely, we add morphism equalities as a third kind of Mmt toplevel

declaration in addition to theories and morphisms. To our knowledge, that yields the first formal system for categories of theories in which users can state and prove the equality of arbitrary morphisms.

We apply this language, in Sect. 5, to develop a pattern for formalizing realms. Concretely, we formalize the realms of lattices and topological spaces. This shows that MMT with morphism equality can serve as a lightweight formalism for realms, and we anticipate this formalism to be more practical than the more involved definition of [CFK14].

A major technical hurdle was that many intuitively true morphism equalities do not actually hold on the nose but depend on the choice of equality. Therefore, to support practical morphism equalities, we first extend LF in Sect. 3 in a way that allows flexibly choosing what logic-specific equality to consider.

We begin by introducing MMT and LF in Sect. 2.

2 Preliminaries

2.1 LF-Expressions

We use LF [HHP93] as a logical framework for defining the logics, in which we state the theories. This is a dependently-typed λ-calculus whose expressions are the universes $\texttt{type}$ and $\texttt{kind}$, typed variables x, typed or kinded constants c, dependent function types $\Pi x : A.B$, abstraction $\lambda x : A.t$, and function application $t\,t'$:

$$E, A, B, s, t ::= x \mid c \mid \lambda x : A.t \mid \Pi x : A.B \mid t\,t \mid \texttt{type} \mid \texttt{kind}$$

Theories Σ declare constants $c : A$ where A is a type or kind. Contexts Γ declare variables $x : A$ where A is a type. The type/kind of a constant and the type of a variable may refer to previously declared constants resp. variables. We usually use A, B as meta-variables for types, s, t for typed terms, E for arbitrary expressions, and we write $A \to B$ for $\Pi x : A.B$ if x does not occur in B and $E[x/t]$ for the substitution of t for x in E.

The judgments are typing $\Gamma \vdash_\Sigma t : A$ and equality $\Gamma \vdash_\Sigma E \overset{expr}{=} E'$. The rules are standard, and we give the rules for expressions in Fig. 1.

*Example 1 (*Typed First-Order Logic*).* We sketch the definition of typed first-order logic FOL as an LF-theory. This representation is routine [HHP93].

$$\begin{array}{lcl} o & : & \texttt{type} \\ \texttt{pf} & : & o \to \texttt{type} \\ \Leftrightarrow & : & o \to o \to o \\ \forall & : & (\Pi A : \texttt{tp}.\texttt{tm}\,A) \to o \end{array} \qquad \begin{array}{lcl} \texttt{tp} & : & \texttt{type} \\ \texttt{tm} & : & \texttt{tp} \to \texttt{type} \\ \overset{FOL}{=} & : & \Pi A : \texttt{tp}.\texttt{tm}\,A \to \texttt{tm}\,A \to o \\ & \ldots & \end{array}$$

Here o is the type of proposition, $\texttt{pf}\,F$ is the type of proofs of proposition F, tp is the LF-type of FOL-types, and $\texttt{tm}\,a$ is the LF-type of FOL-terms of FOL-type a. Note that FOL provides its own equality *connective* $\overset{FOL}{=}$, which is different

$$\frac{}{\Gamma \vdash_\Sigma \mathtt{type} : \mathtt{kind}} \qquad \frac{U \in \{\mathtt{type}, \mathtt{kind}\} \quad \Gamma, x : A \vdash_\Sigma E : U}{\Gamma \vdash_\Sigma \Pi x : A.E : U}$$

$$\frac{U \in \{\mathtt{type}, \mathtt{kind}\} \quad \Gamma \vdash_\Sigma E : U \quad \Gamma, x : A \vdash_\Sigma t : E}{\Gamma \vdash_\Sigma \lambda x : A.t : \Pi x : A.E} \qquad \frac{\Gamma \vdash_\Sigma f : \Pi x : A.E \quad \Gamma \vdash_\Sigma t : A}{\Gamma \vdash_\Sigma f\, t : E[x/t]}$$

$$\frac{}{\Gamma \vdash_\Sigma E \overset{expr}{=} E} \qquad \frac{Q \in \{\lambda, \Pi\} \quad \Gamma \vdash_\Sigma A \overset{expr}{=} A' \quad \Gamma, x : A \vdash_\Sigma E \overset{expr}{=} E'}{\Gamma \vdash_\Sigma Qx : A.E \overset{expr}{=} Qx : A'.E'}$$

$$\frac{\Gamma \vdash_\Sigma E \overset{expr}{=} E' \quad \Gamma \vdash_\Sigma F \overset{expr}{=} F'}{\Gamma \vdash_\Sigma E\, F \overset{expr}{=} E'\, F'}$$

$$\frac{}{\Gamma \vdash_\Sigma (\lambda x : A.E)F \overset{expr}{=} E[x/F]} \qquad \frac{\Gamma \vdash_\Sigma f : \Pi x : A.B}{\Gamma \vdash_\Sigma \lambda x : A.(f\, x) \overset{expr}{=} f}$$

Fig. 1. Typing and Equality Rules of LF

from the LF-*judgment* $\overset{expr}{=}$. We use higher-order abstract syntax for binders (e.g., $\forall\, A\, (\lambda x : i.F)$ represents the proposition $\forall x : A.F$) and curried functions for the connectives (e.g., $\Leftrightarrow\ F\, G$ represents $F \Leftrightarrow G$), and we will use the common notations in the sequel for those expressions.

2.2 Theory Graphs in Mmt

The grammar of the instantiation of Mmt with LF is given in Fig. 2, where A, E, t are LF-expressions as above and $\emptyset$ denotes the empty theory graph.

$$\begin{aligned}
G &::= \emptyset \mid G,\ \mathbf{theory}\ s = \{\sigma\} \mid G,\ \mathbf{theory}\ s = S \\
&\quad \mid G, \mathbf{morph}\ m : S \longrightarrow T = \{\mu\} \mid G, \mathbf{morph}\ m : S \longrightarrow T = M \\
\sigma &::= D^* \qquad D ::= c : A \mid \mathbf{include}\ S\ [= M] \\
\mu &::= d^* \qquad d ::= c[: A] = E \mid \mathbf{include}\ S = M \\
S &::= s \mid S \cup S \\
M &::= m \mid id_S \mid M; M \mid M \cup M
\end{aligned}$$

Fig. 2. Mmt Grammar

A theory graph G consists of theory and morphism declarations. A **primitive theory declaration theory** $s = \{\sigma\}$ introduces the theory named s given by the list of declarations σ. The declarations in the body of a theory are of the form $c : A$ where c is a name and A is the type/kind of c. Alternatively, we can introduce a **defined theory** by **theory** $s = S$, which defines s as an abbreviation for a theory expression S. **Theory expressions** S are either references s to theory names or unions $S \cup T$ of theories.

meta-vars.	thy.	morph.
name	s	m
expression	S	M
body	σ	μ

The syntax of morphisms is analogous to that of theories. The names we use for meta-variables are summarized on the right. A **primitive morphism declaration morph** $m : S \longrightarrow T = \{\mu\}$ introduces the morphism named m from S to T given by the list of declarations μ. The declaration is well-formed if μ contains exactly one declaration $c[: A'] = E$ for every constant $c : A$ of S such that $E : m(A)$ holds over T. (The type A' is redundant and must be equal to $m(A)$ if given.) m induces a compositional type-preserving mapping $m(-)$ of S-expressions to T-expressions by homomorphic extension, i.e., by replacing every constant with the image provided by μ.

Alternatively, we can introduce **defined morphisms** by **morph** $m : S \longrightarrow T = M$ for a morphism expression M. **Morphism expressions** M are references m to morphism names, identity morphisms $id_S : S \longrightarrow S$, compositions $M; N : R \longrightarrow T$ of $M : R \longrightarrow S$ and $N : S \longrightarrow T$, or unions $M_1 \cup M_2 : S_1 \cup S_2 \longrightarrow T$ of $M_1 : S_1 \longrightarrow T$ and $M_2 : S_2 \longrightarrow T$. Every morphism expression M defines a compositional homomorphic mapping $M(-)$ given by, respectively, $m(-)$, the identity map, the composition of $M(-)$ and $N(-)$, and the union of $M_1(-)$ and $M_2(-)$. In particular, we have $(M_1 \cup M_2)(E) = M_i(E)$ if E is an S_i-expression.

In addition to the above, a primitive theory or morphism may contain **include declarations**. In a theory with name t, the declaration **include** $S[= M]$ reuses all constants of S for t. A recent and previously unpublished feature of Mmt that will prove critical for our formalizations is that such includes may carry a definiens $M : S \longrightarrow t$. Defined includes can be seen as analogous to defined constants: from the perspective of t, (i) a morphism $M : S \longrightarrow t$ can be seen as an object of "type" S, (ii) an include of S specifies that t is a subtype of S, and (iii) a definiens M specifies that t can be viewed as an instance of S via M. In terms of object-oriented programming an undefined include is inheritance of S into t, and a defined include is delegation from t to M for interface S. Similarly, in a primitive morphism $m : S \longrightarrow T$, the declaration **include** $R = N$ for a morphism $N : R \longrightarrow T$ reuses all mappings of N, i.e., we have $m(c) = N(c)$ for every R-constant c.

Example 2. We spell out our running example in Mmt syntax in Fig. 3. We omit the axioms for brevity and only remark that axioms are treated in the same way as constants: they are declared as constants (of type `pf` F for some F) and mapped by morphisms to appropriate `FOL`-proof terms. Note how the morphisms include $id_{\texttt{Carrier}}$. This makes explicit that, e.g., `GtoDG` is equal to the identity when restricted to the smaller domain `Carrier`.

Relative to a theory graph G, the type system uses **judgments** given in Fig. 4. Due to include declarations, the semantics of a theory now depends on the entire theory graph. Therefore, we have to index the `LF`-judgments for expressions with G as well.

Fig. 5 gives the most important rules, which we explain in the remainder. But before doing so, we state the **main theorem** about Mmt to solidify the

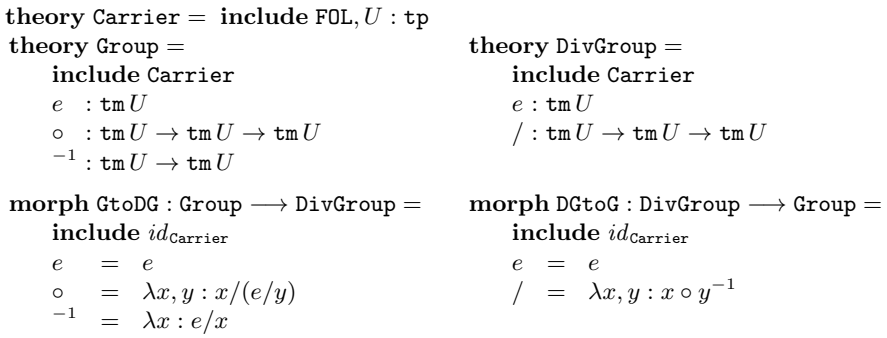


Fig. 3. Example Theory Graph in MMT

Judgment	Intuition
$\Gamma \vdash^G_T E : E'$	typing of LF-expression over theory T
$\Gamma \vdash^G_T E \stackrel{expr}{=} E'$	equality of LF-expressions over theory T
$\vdash^G T$ THY	well-formed theory expression
$\vdash^G S \stackrel{M}{\hookrightarrow} T$	S included into T ($M = id_S$ if omitted)
$\vdash^G M : S \longrightarrow T$	well-formed morphism expression
$\vdash^G M \stackrel{\text{mor}}{=} N : S \longrightarrow T$	morphism equality

Fig. 4. MMT Judgments

intuition of morphisms: they preserve all judgments, i.e., the following rules are admissible

$$\frac{\Gamma \vdash^G_T t : A \qquad \vdash^G M : S \longrightarrow T}{M(\Gamma) \vdash^G_S M(t) : M(A)} \qquad \frac{\Gamma \vdash^G_T t \stackrel{expr}{=} t' \qquad \vdash^G M : S \longrightarrow T}{M(\Gamma) \vdash^G_S M(t) \stackrel{expr}{=} M(t')}$$

The rules for **well-formed theories** are straightforward. Technically, we need an equality judgment for theory expressions here with rules for definition expansion and idempotence, commutativity, associativity of union, but we omit that for brevity.

The rules for the **inclusion judgment** $\vdash^G S \stackrel{M}{\hookrightarrow} T$ build the category generated by the **include** declarations in theories. The morphism M is optional, and if it is omitted, we assume $M = id_S$. Its intuition is formalized in the rule *Lookup*, which makes all constants from an included theory available to the including theory. Consider a declaration **include** S in a primitive theory t. Then we have $\vdash^G S \stackrel{id_S}{\hookrightarrow} t$, and *Lookup* makes any declaration $c : A$ of S available to T unchanged. The second conclusion of *Lookup* vacuously establishes $c \stackrel{expr}{=} c$. Alternatively, consider a declaration **include** $S = M$ for $\vdash^G M : S \longrightarrow t$. Now $\vdash^G S \stackrel{M}{\hookrightarrow} t$, and *Lookup* makes the declaration $c : M(A)$ available to T, and its

$$\frac{\textbf{theory}\ t = \{\sigma\}\ \text{in}\ G \qquad c : A\ \text{in}\ \sigma \qquad \vdash^G t \overset{M}{\hookrightarrow} T}{\vdash^G_T c : M(A) \quad \text{and} \quad \vdash^G_T c \overset{expr}{=} M(c)}\ \mathit{Lookup}$$

$$\frac{\textbf{theory}\ t = \_\ \text{in}\ G}{\vdash^G t\ \mathtt{THY}} \qquad \frac{\vdash^G S\ \mathtt{THY} \qquad \vdash^G T\ \mathtt{THY}}{\vdash^G S \cup T\ \mathtt{THY}}$$

$$\frac{\textbf{theory}\ t = \{\sigma\}\ \text{in}\ G \qquad \textbf{include}\ S[= M]\ \text{in}\ \sigma}{\vdash^G S \overset{[M]}{\hookrightarrow} t} \qquad \frac{}{\vdash^G T \hookrightarrow T} \qquad \frac{\vdash^G R \overset{M}{\hookrightarrow} S \qquad \vdash^G S \overset{N}{\hookrightarrow} T}{\vdash^G R \overset{M;N}{\hookrightarrow} T}$$

$$\frac{\textbf{morph}\ m : S \longrightarrow T = \_\ \text{in}\ G}{\vdash^G m : S \longrightarrow T} \qquad \frac{\vdash^G M : S \longrightarrow T \qquad \vdash^G R \hookrightarrow S \qquad \vdash^G T \hookrightarrow U}{\vdash^G M : R \longrightarrow U}\ \mathit{compIncl}$$

$$\frac{\vdash^G M : R \longrightarrow S \qquad \vdash^G N : S \longrightarrow T}{\vdash^G M;N : R \longrightarrow T} \qquad \frac{}{\vdash^G S_1 \hookrightarrow S_1 \cup S_2} \qquad \frac{}{\vdash^G S_2 \hookrightarrow S_1 \cup S_2}$$

$$\frac{\vdash^G S_1 \overset{M_1}{\hookrightarrow} T \qquad \vdash^G S_2 \overset{M_2}{\hookrightarrow} T \qquad \vdash^G M_1 \overset{\text{mor}}{=} M_2 : S_1 \cap S_2 \longrightarrow T}{\vdash^G S_1 \cup S_2 \overset{M_1 \cup M_2}{\hookrightarrow} T}$$

$$\frac{\vdash^G M_1 : S_1 \longrightarrow T \qquad \vdash^G M_2 : S_2 \longrightarrow T \qquad \vdash^G M_1 \overset{\text{mor}}{=} M_2 : S_1 \cap S_2 \longrightarrow T}{\vdash^G M_1 \cup M_2 : S_1 \cup S_2 \longrightarrow T}$$

Fig. 5. Typing Rules for Theory Graphs

second conclusion makes c an abbreviation for $M(c)$. Thus, defined includes are always conservative and just add defined constants.

The rules for **well-formed morphisms** build the category generated by the named morphisms. If $\vdash^G S \hookrightarrow T$, we do not introduce a name for the induced embedding of S-expressions into T-expressions; instead, rule *compIncl* allows composing morphisms with inclusions. In particular, if $\vdash^G S \hookrightarrow T$, we have $\vdash^G id_S : S \longrightarrow T$.

The judgment for **morphism equality** comes in critically in the two rules in Fig. 5 that involve morphisms out of a union theory. For example, the rule for the morphism union $M_1 \cup M_2 : S_1 \cup S_2 \longrightarrow T$ requires that the M_i agree on the intersection of their domains. Formally, we define $S_1 \cap S_2$ as the union of all named theories t that are included without definition into both S_i, i.e., all t for which $\vdash^G t \hookrightarrow S_1$ and $\vdash^G t \hookrightarrow S_2$. Then to say that the M_i agree on $S_1 \cap S_2$ means that $\vdash^G M_1 \overset{\text{mor}}{=} M_2 : t \longrightarrow T$ for every such t.

Morphism equality is also critical in the well-formedness of include declarations. Include declarations in theories are only well-formed if for any S, T, there is at most one M such that $\vdash^G S \overset{M}{\hookrightarrow} T$, i.e., theories must not be included via two different morphisms. Similarly, in a morphism m, include declarations are only well-formed if no two different morphisms are included for the same theory. In both cases, the formal condition checked by MMT is that the declarations **include** $S_1 = M_1$ and **include** $S_2 = M_2$ may only occur together in the same primitive theory/morphism if $\vdash^G M_1 \overset{\text{mor}}{=} M_2 : S_1 \cap S_2 \longrightarrow T$, where T is the containing theory or, respectively, the codomain of the containing morphism.

Fig. 5 omits the rules for establishing morphism equality. Generally, two morphisms are equal if they induce the same homomorphic mapping, which is equivalent to mapping every constant of the domain to equal expressions. But even if the equality of expressions is decidable (as for LF), this is a far too expensive criterion in practice — morphism equality must be checked very frequently, and each time an expression equality check would be needed for every domain constant. Therefore, MMT uses an incomplete sufficient criterion that implements diagram chase–reasoning without ever inspecting the bodies of primitive morphisms. We defer the presentation to Sect. 4, where we change the rules anyway.

3 Propositional Equality of LF-Expressions

Example 3 (Failure of Morphism Equality). To prove

$$\vdash^G \texttt{DGtoG}; \texttt{GtoDG} \overset{\text{mor}}{=} id_{\texttt{DivGroup}} : \texttt{DivGroup} \longrightarrow \texttt{DivGroup}$$

we must show that both morphisms map each constant to equal expressions, e.g., we need the equality of $\texttt{GtoDG}(\texttt{DGtoG}(/)) = \texttt{GtoDG}(\lambda x, y.x \circ y^{-1}) = \lambda x, y.x/y^{-1^{-1}}$ and $id_{\texttt{DivGroup}}(/) = /$ (where we have silently applied the necessary β-reductions). But these terms are only *provably equal* in the FOL-theory DivGroup. LF, which only uses $\alpha\beta\eta$-equality, does *not* consider them equal.

Ex. 3 shows that morphism equality cannot easily be defined generically at the MMT- or LF-level because it may depend on logic-specific equalities. For example, FOL-constants can be type, function, predicate symbols, or axioms, and FOL does not support equality for any of them out of the box. Consider functions f, g of type $\texttt{tm}\, U \to \texttt{tm}\, U$. The natural choice for equality is the formula $\forall x : \texttt{tm}\, U. f\, x \overset{FOL}{=} g\, x$. For predicates $p, q : \texttt{tm}\, U \to o$, it would be $\forall x : \texttt{tm}\, U. p\, x \Leftrightarrow q\, x$. For types, FOL does not provide any equality, and we have to fall back to LF-equality. For axioms, the simplest choice is a proof irrelevance rule, where any expressions $P, Q : \texttt{pf}\, F$ are considered equal.

Our key idea is to define LFQ by adding a propositional equality predicate to LF that logic developers can use to spell out these equalities, so that MMT can consider them when checking the equality of two morphisms.

The idea of adding propositional equality to LF is not new. One approach is to add rewriting as in Dedukti [CD07]. Another option is to add identity types as in Martin-Löf type theory [ML74]. Our formulation below is essentially the same as the one worked out in [Har21].

We add a kind $E \overset{LF}{=}_A E'$ for the equality of terms E and E' of type A. We could make this a type, but that would amount to using identity types and be much more expressive than needed for our purposes. Because LF can quantify over types but not over kinds, $E \overset{LF}{=}_A E'$ can only occur as the output of LF-constants but not as input. Thus, users can declare new propositional equalities but can never do anything with them — it remains the discretion of the

system how to use them. That is important because user-declared propositional equalities make typing in LFQ undecidable, and implementations will only be able to handle them to a limited degree.

The LFQ grammar extends the one of LF with

$$E ::= E \overset{LF}{=}_A E \mid \texttt{refl} \mid \texttt{funExt}\, E$$

Note that we now distinguish the *kind* $E \overset{LF}{=}_A E$ for equality of typed terms and the *judgment* $\vdash E \overset{expr}{=} E'$ for the equality of expressions. LFQ adds the following rules to LF

$$\frac{\Gamma \vdash^G_T A : \texttt{type} \quad \Gamma \vdash^G_T E : A \quad \Gamma \vdash^G_T E' : A}{\Gamma \vdash^G_T (E \overset{LF}{=}_A E') : \texttt{kind}}$$

$$\frac{\Gamma \vdash^G_T P : \Pi x_1 : A_1, \ldots, x_n : A_n.(E\, x_1\ \ldots\ x_n \overset{LF}{=}_B E'\, x_1\ \ldots\ x_n)}{\Gamma \vdash^G_T \texttt{funExt}\, P : (E \overset{LF}{=}_{\Pi x_1:A_1,\ldots,x_n:A_n.B} E')}$$

$$\frac{\Gamma \vdash^G_T P : (E \overset{LF}{=}_A E')}{\Gamma \vdash^G_T E \overset{expr}{=} E'} \qquad \frac{\Gamma \vdash^G_T A : \texttt{type} \quad \Gamma \vdash^G_T E : A}{\Gamma \vdash^G_T \texttt{refl} : (E \overset{LF}{=}_A E)}$$

The first rule enables users to declare new propositional equalities. The second allows using funExt to show the equality of two functions by functional extensionality. The other two rules map back and forth between the judgment $\overset{expr}{=}$ and the kind $\overset{LF}{=}$.

From now on, we work in the instantiation of Mmt with LFQ. Because Mmt allows the modular definition of logical frameworks, and LFQ only adds constructors and rules to LF, any LF-theory graph is also an LFQ theory graph.

*Example 4 (*FOL*-Specific Equality).* We extend FOL from Ex. 1 to the logic FOLQ in LFQ by adding propositional equalities that quotient FOL-expressions:

theory FOLQ =

include FOL

$\texttt{eqT} : \Pi A : \texttt{tp}.\, \Pi x, y : \texttt{tm}\, A.(\texttt{pf}\, x \overset{FOL}{=} y) \to x \overset{LF}{=}_{\texttt{tm}\, A} y$

$\texttt{eqF} : \Pi f, g : o.(\texttt{pf}\, f \Leftrightarrow g) \to f \overset{LF}{=}_o g$

$\texttt{eqP} : \Pi f : o.\, \Pi p, q : \texttt{pf}\, f.\, p \overset{LF}{=}_{\texttt{pf}\, f} q$

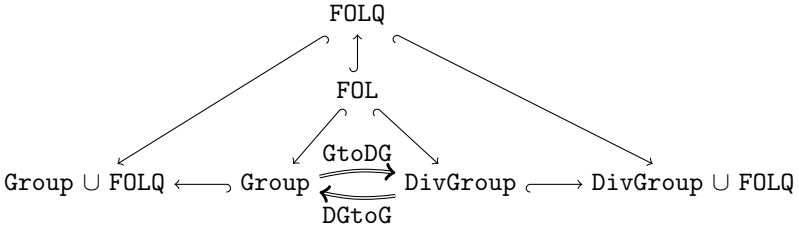
eqT makes terms LF-equal if they are provably equal in FOL. Using functional extensionality, this implies, e.g., for two unary functions $f, g : \texttt{tm}\, U \to \texttt{tm}\, U$

$$\vdash f \overset{LF}{=}_{\texttt{tm}\, U \to \texttt{tm}\, U} g \qquad \text{iff} \qquad x : \texttt{tm}\, U \vdash P : \texttt{pf}(f\, x \overset{FOL}{=} g\, x)$$

The constant eqF does the same for formulas and, e.g., for unary predicates $p, q : \texttt{tm}\, A \to o$. The constant eqP adds proof irrelevance.

FOLQ injects its undecidable equality into LFQ, thus rendering $\vdash E \overset{expr}{=} E'$ undecidable. But the deep research problems associated with that go way beyond the purpose of this paper. Instead, our plan is to use FOLQ only as the codomain of morphism equality judgments, in which case the undecidability is manageable:

Example 5 (Morphism Equality via a Stronger Codomain). Consider the theory graph below that summarizes our running example



The judgment $\vdash^G \texttt{DGtoG};\texttt{GtoDG} \overset{\text{mor}}{=} id_{\texttt{DivGroup}} : \texttt{DivGroup} \longrightarrow \texttt{DivGroup} \cup \texttt{FOLQ}$ holds. Here the same morphisms as in Ex. 3 are compared relative to a bigger codomain in which additional propositional equalities are declared. Thus, the resulting proof obligations (which are equalities of expressions over the codomain) are checked relative to a stronger theory.

Indeed, we have a FOL-proof

$$x : \texttt{tm}\, U,\, y : \texttt{tm}\, U \vdash_{\texttt{DivGroup}\cup\texttt{FOLQ}} I : \texttt{pf}\big(x/(y^{-1^{-1}}) \overset{FOL}{=} x/y\big)$$

which we can use to show $\vdash_{\texttt{DivGroup}\cup\texttt{FOLQ}} \lambda x, y.x/y^{-1^{-1}} \overset{expr}{=} /$. The corresponding cases for the other constants of DivGroup as well as for the dual equality of $\texttt{GtoDG};\texttt{DGtoG}$ and $id_{\texttt{Group}}$ can be shown accordingly.

Thus, Group and DivGroup are not isomorphic in the category of theories that include FOL, but $\texttt{Group} \cup \texttt{FOLQ}$ and $\texttt{DivGroup} \cup \texttt{FOLQ}$ are isomorphic in the category of theories that include FOLQ.

Ex. 5 shows that we can model different equality relations on morphisms by using different codomains. This is extremely valuable because it keeps the formalism simple by retaining a single equality judgment and uses the modularity of the theory graph to capture different levels of equality.

4 Propositional Equality of MMT-Morphisms

It remains to extend the MMT language in a way that can utilize the propositional *expression* equality introduced in Sect. 3 to prove *morphism* equalities. We extend the grammar as below and explain all new productions in the remainder:

$$\begin{aligned} G ::= \ldots \mid\ & G, \textbf{morpheq}\ k : M \overset{\text{mor}}{=} N : S \longrightarrow T = \{\kappa\} \\ \mid\ & G, \textbf{morpheq}\ k : M \overset{\text{mor}}{=} N : S \longrightarrow T = K \\ \kappa ::= d^* \qquad & d ::= c[: A] = E \mid \textbf{include}\ S = K \end{aligned}$$

$$K ::= k \mid \texttt{refl}\, M \mid \text{(other proof terms)}$$

A **morphism equality declaration** is a theorem named k stating the equality of two morphisms M and N, both from S to T. In the **primitive** case, k is proved by giving a body κ. Just like the body σ of a primitive theory t gives the constructors of t-expressions, and the body μ of a primitive morphism m with domain t gives the cases of a compositional mapping of t-expressions, the body κ of a primitive morphism equality gives the cases of the inductive equality proof for two such morphisms.

A primitive morphism equality **morpheq** $k : M \overset{\text{mor}}{=} N : s \longrightarrow T = \{\kappa\}$, where s is a primitive theory with body σ, is well-formed if:

- For every constant $c : A$ in σ, κ contains exactly one $c[: A'] = E$ where $\vdash^G_T E : (M(c) \overset{LF}{=}_{M(A)} N(c))$. (The expression A' is redundant. If given, it must be equal to the type of E.)
- For every **include** R in σ, κ contains exactly one **include** $R = K$ where K is a proof term for $\vdash^G R \overset{\text{mor}}{=} T : M \longrightarrow N$.

If the domain of k is a union theory $S_1 \cup S_2$, κ must provide cases for the declarations of each S_i. If it is a *defined* named theory, we expand the definiens first and apply the definition above.

Example 6. We show one of the two isomorphism properties of our example:

$$\begin{array}{l}
\textbf{morpheq}\ k : \texttt{DGtoG}; \texttt{GtoDG} \overset{\text{mor}}{=} id_{\texttt{DivGroup}} : \texttt{DivGroup} \longrightarrow \texttt{DivGroup} \cup \texttt{FOLQ} = \\
\quad \textbf{include}\ \texttt{Carrier} = \texttt{refl}\ id_{\texttt{Carrier}} \\
\quad e : \ e \overset{LF}{=}_{\texttt{tm}\,U}\ e \ = \ \texttt{refl} \\
\quad / : \ \lambda x, y.x/(y^{-1^{-1}}) \overset{LF}{=}_{\texttt{tm}\,U \to \texttt{tm}\,U \to \texttt{tm}\,U} / \ = \\
\qquad\qquad \texttt{funExt}\, \lambda x, y.\, \texttt{eqT}\, U\, (x/(y^{-1^{-1}}))\, (x/y)\, I
\end{array}$$

Both morphisms restrict to $id_{\texttt{Carrier}}$ on the theory `Carrier`. Consequently, we use a reflexivity proof for $\vdash^G id_{\texttt{Carrier}} \overset{\text{mor}}{=} id_{\texttt{Carrier}} : \texttt{Carrier} \longrightarrow \texttt{DivGroup} \cup \texttt{FOLQ}$. In the declaration for e, we have $(\texttt{DGtoG}; \texttt{GtoDG})(e) = e = id_{\texttt{DivGroup}}(e)$ so that the reflexivity proof for `LF` expressions suffices. In practical implementations, those two cases could be omitted and filled in by the system as defaults. Finally, the declaration for / discharges the proof obligation that failed in Ex. 3 using the proof I from Ex. 5.

If we had not omitted the axiom declarations from `DivGroup`, we would also have to show the equality of the proofs assigned to the axioms. That would be trivial due to the use of proof irrelevance in `FOLQ`.

In the **defined** case **morpheq** $k : M \overset{\text{mor}}{=} N : s \longrightarrow T = K$, we require that K is a proof term for the morphism equality judgment $\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T$. Originally, we wanted to support only the primitive case. However, our case studies showed that, apart from making the syntax of theories, morphisms, and equalities analogous, the defined case is critically important in practice. Because propositional equality is undecidable but must be called frequently, practical implementations must employ cheap incomplete heuristics instead of running a

theorem prover to discharge a morphism equality. But incompleteness threatens scalability — it is imperative that users are able to workaround situations where the system runs into a proof obligation $\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T$ that it cannot prove. We found defined morphism equalities to be the right compromise here: if a morphism equality is implied by the given primitive morphism equalities but the system cannot find the proof, the user can give a defined morphism equality to show the proof to the system. Because K is a diagram chase-style proof term, that is orders of magnitude easier than proving a new primitive morphism equality. We give an example in Sect. 5.

For brevity, our grammar omits the productions for **morphism equality proof terms** K. They arise as the straightforward proof term assignment to the inference system for the judgment $\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T$, whose rules we give now. The key rules are

$$\frac{\textbf{morpheq}\ k : M \overset{\text{mor}}{=} N : S \longrightarrow T = \_ \text{ in } G}{\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T}\,base$$

$$\frac{\textbf{morph}\ m : S \longrightarrow T = M \text{ in } G}{\vdash^G m \overset{\text{mor}}{=} M : S \longrightarrow T}\,def$$

$$\frac{\textbf{morph}\ m : S \longrightarrow T = \{\mu\} \text{ in } G \qquad \textbf{include}\ R = L \text{ in } \mu}{\vdash^G m \overset{\text{mor}}{=} L : R \longrightarrow T}\,morphIncl$$

$$\frac{\vdash^G M : S \longrightarrow T \qquad \vdash^G R \hookrightarrow S \qquad \vdash^G M \overset{\text{mor}}{=} N : R \longrightarrow T}{\vdash^G M \overset{\text{mor}}{=} M \cup N : S \longrightarrow T}\,unionIncl$$

The first two rules are straightforward: *base* gives the base case of equalities explicitly proved by the user, and *def* expands the definition of defined morphisms. *morphIncl* gives the semantics of **include** $R = L$ in a primitive morphism m: L is the restriction of m to R. *unionIncl* is a subsumption rule that allows removing redundant parts in a union of morphisms.

The remaining rules are routine, and we only sketch them for brevity:

- equivalence (reflexivity, symmetry, transitivity) and congruence (substitution of equals by equals) of morphism equality
- category axioms (associativity of composition, neutrality of identity)
- semilattice properties of union (idempotence, commutativity, associativity)

Finally, we can obtain the **main theorem** that captures the soundness of the morphism equality calculus: equal morphism induce equal expression mappings, i.e., the following rule is admissible

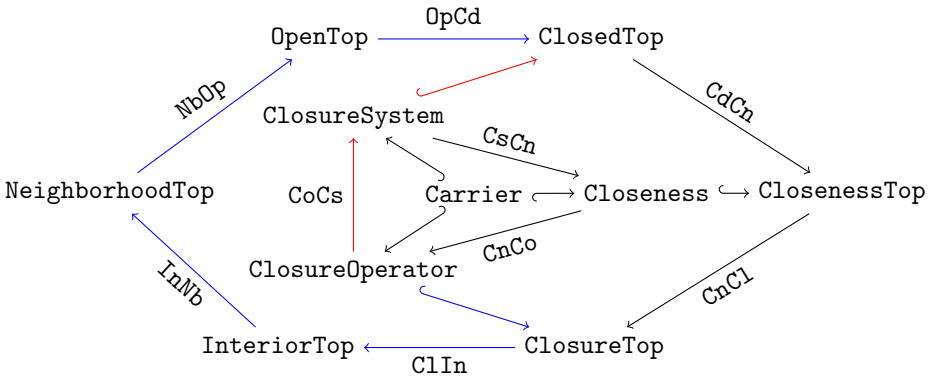
$$\frac{\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T \qquad \Gamma \vdash^G_S t : A}{M(\Gamma) \vdash^G_T M(t) \overset{expr}{=} N(t)}$$

It is proved by induction on the derivations of $\vdash^G M \overset{\text{mor}}{=} N : S \longrightarrow T$.

5 Case Studies

With morphism equality in place, we can now finish the two case studies that we had to leave incomplete in [RW22].[2] Both are essentially the same as in [RW22]. But we are now able to state and prove the various morphism equalities.

Topological Spaces There are many isomorphic definitions of topological space. Even more interestingly, many of them extend closure systems, for which there are also multiple isomorphic definitions. Concretely, our formalization consists of three isomorphic theories for closure systems and six isomorphic theories for topological spaces as shown in the theory graph below. Here the inner triangle and the outer rectangle are isomorphism cycles.



The bodies of these theories and morphisms are inessential for our purposes here. For example, `ClosureSystem` uses an intersection-closed set of subsets of the carrier set whereas `ClosureOperator` uses an idempotent mapping on subsets.

Crucially, the whole theory graph commutes. In particular, to show the isomorphisms, we have proved three primitive morphism equalities to show that the inner triangle commutes and six to show that the outer hexagon commutes. For example, we prove **morpheq** `isoClosureSystem` : `CsCn;CnCo;CoCs` $\overset{\text{mor}}{=}$ $id_{\texttt{ClosureSystem}}$: `ClosureSystem` $\longrightarrow$ `ClosureSystem` $= \{\ldots\}$. While we have not fully implemented morphism equality in Mmt yet, all proofs in the bodies of these morphism equalities were done in and checked by Mmt.

The commutativity of the rectangles connecting the inner with the outer ring hold definitionally: for example, **include** `ClosureSystem` = `CsCn` is contained in the body of `CdCn`, at which point the rule *morphIncl* yields $\vdash^G$ `CdCn` $\overset{\text{mor}}{=}$ `CsCn` : `ClosureSystem` $\longrightarrow$ `ClosenessTop`. Similarly, all edges of the inner triangle include $id_{\texttt{Carrier}}$, which makes the triangles involving `Carrier` commute.

The only rectangle whose commutativity requires a non-trivial proof term is $\vdash^G$ `ClIn;InNb;NbOp;OpCd` $\overset{\text{mor}}{=}$ `CoCs` : `ClosureOperator` $\longrightarrow$ `ClosedTop` $(*)$. This equality follows from the other morphism equalities mentioned above by diagram chase, i.e., by applying the rules given in Sect. 4, mostly tedious uses

[2] Both (as well as our running example) are available at https://gl.mathhub.info/MMT/LATIN2/-/tree/devel/source/casestudies/2023-morpheq.

of associativity and substitution. A concrete implementation of this undecidable property may or may not manage to discharge (∗) automatically and swiftly. If it fails, users can state a defined morphism equality to work around this incompleteness.

To integrate all the isomorphic theories into a single realm in the sense of [CFK14], we use MMT's defined includes as follows:

theory `Closure` =
  include `ClosureSystem`
  include `ClosureOperator` = `CoCs`
  include `Closeness` = `CnCo`

theory `Topology` =
  include `Closure`
  include `ClosedTop`
  include `OpenTop` = `OpCd`
  include `NeighborhoodTop` = `NbOp`
  include `InteriorTop` = `InNb`
  include `ClosureTop` = `ClIn`
  include `ClosenessTop` = `CnCl`

Here `Topology` can use all operations from any one of the six isomorphic theories because they are all included. Critically, the definitions of the includes ensure that all six includes refer to the same underlying topology. For example, **include** `ClosureTop` = `ClIn` also includes the theory `ClosureOperator`, which has already been included via `Closure`. Thus, checking the well-formedness of `Topology` generates the proof obligation (∗). Previously, MMT could not discharge (∗), and users had no way to help it along.

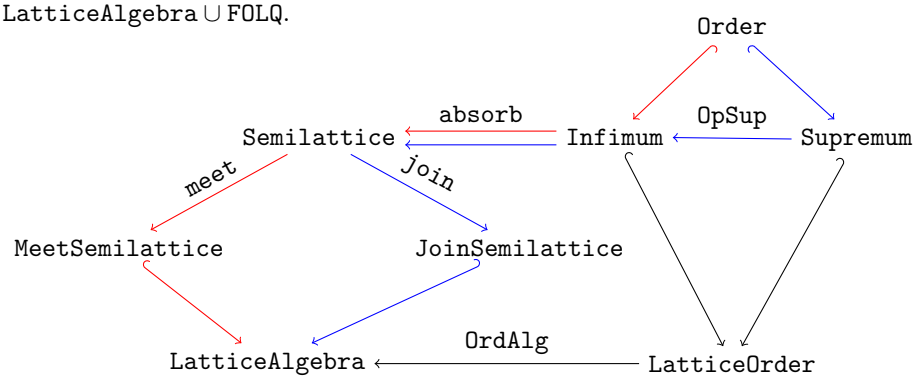
Lattices We give two isomorphic formalizations of lattices in the theory graph below: Firstly, `LatticeAlgebra` is based on two copies of `Semilattice` (with operation ∘) given by the two morphisms `meet` (mapping ∘ to ⊓) and `join` (mapping ∘ to ⊔). Secondly, `LatticeOrder` is based on an order ≤ and arises as the union of `Infimum` and `Supremum`. Even just giving the morphism `OrdAlg` (without even trying to prove it to be an isomorphism) was previously impossible in MMT.

The issue is subtle. The isomorphism `absorb` defines an infimum relation for every semilattice by mapping $\leq = \ \lambda x, y.x \circ y \overset{FOL}{=} x$. By composing it with `meet`, we obtain the infimum operation in algebraic lattices. Correspondingly, we obtain the supremum by composing it with `join` and `OpSup` (which maps $\leq = \ \lambda x, y.y \leq x$).

Thus, `OrdAlg` can be defined elegantly using **include** `Infimum` = `absorb;meet` and **include** `Supremum` = `OpSup;absorb;join`. These two morphisms now have to agree on `Infimum` ∩ `Supremum` = `Order`, i.e., we have the morphism equality proof obligation $\vdash^G$ (`absorb;meet`) $\overset{\text{mor}}{=}$ (`OpSup;absorb;join`) : `Order` ⟶ `LatticeAlgebra`. That in turn generates the expression equality proof obligation $\vdash_{\texttt{LatticeAlgebra}} (x \sqcap y \overset{FOL}{=} x) \overset{expr}{=} (y \sqcup x \overset{FOL}{=} y)$ (∗). But this holds in `LatticeAlgebra` only up to ⇔.

We can remedy this by proving a morphism equality **morpheq** `ordersAgree` : (`absorb;meet`) $\overset{\text{mor}}{=}$ (`OpSup;absorb;join`) : `Order` ⟶ `LatticeAlgebra`∪`FOLQ` = {≤ = `funExt` $\lambda x, y.$ `eqF`(...)}, where we use `eqF` to discharge (∗). With this

equality in place, `OrdAlg` becomes well-formed as a morphism `LatticeOrder` $\longrightarrow$ `LatticeAlgebra` $\cup$ `FOLQ`.



6 Conclusion and Future Work

We showed how to extend theory graph formalisms with proofs of equality of morphisms. Besides theories and morphisms, morphism equality is the third constitutive component of categorical diagrams, but it had received little attention in prior work on theory graphs. We showed that even elementary examples of theory graphs, such as the definitions of lattices and topological spaces, require a systematic treatment of morphism equality that had not been done before, and we have shown how our design enables this treatment. We used the Mmt language for theory graphs instantiated with the logical framework `LF` to present our design in a concrete and logic-independent setting, and our ideas carry over easily to other theory graph formalisms. Moreover, by combining our design with defined includes, we have demonstrated a promising formalization pattern for realms, a theory graph formalism feature that had previously been called for [CFK14] but not realized by any practical system.

We are currently implementing our design by extending the Mmt tool for theory graphs. To ensure the feasibility of this, we have taken care to evaluate our approach in multiple case studies. These are already available in the anticipated Mmt syntax, and all proofs in them have already been developed in and verified by Mmt.

References

AHMS99. 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.

CD07. D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In S. Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 102–117. Springer, 2007.

CFK14. J. Carette, W. Farmer, and M. Kohlhase. Realms: A Structure for Consolidating Knowledge about Mathematical Theories. In S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban, editors, *Intelligent Computer Mathematics*, pages 252–266. Springer, 2014.

FGT92. W. Farmer, J. Guttman, and F. Thayer. Little Theories. In D. Kapur, editor, *Conference on Automated Deduction*, pages 467–581, 1992.

FGT93. W. Farmer, J. Guttman, and F. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, 1993.

Har21. R. Harper. An equational logical framework for type theories, 2021. https://arxiv.org/abs/2106.01484.

HHP93. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

KWP99. F. Kammüller, M. Wenzel, and L. Paulson. Locales – a Sectioning Concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics*, pages 149–166. Springer, 1999.

ML74. P. Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Proceedings of the '73 Logic Colloquium*, pages 73–118. North-Holland, 1974.

MML07. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *Tools and Algorithms for the Construction and Analysis of Systems 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.

ORS92. 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.

RK13. F. Rabe and M. Kohlhase. A Scalable Module System. *Information and Computation*, 230(1):1–54, 2013.

RS09. F. Rabe and C. Schürmann. A Practical Module System for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, pages 40–48. ACM Press, 2009.

RW22. F. Rabe and F. Weber. Three Case Studies on Realms. In K. Buzzard and T. Kutsia, editors, *Intelligent Computer Mathematics, Informal Proceedings*, pages 46–51. Research Institute for Symbolic Computation, 2022.

SJ95. Y. Srinivas and R. Jüllig. Specware: Formal Support for Composing Software. In B. Möller, editor, *Mathematics of Program Construction*. Springer, 1995.

SW83. D. Sannella and M. Wirsing. A Kernel Language for Algebraic Specification and Implementation. In M. Karpinski, editor, *Fundamentals of Computation Theory*, pages 413–427. Springer, 1983.