

A Foundational View on Integration Problems

Florian Rabe¹ and Michael Kohlhase¹ and Claudio Sacerdoti Coen²

¹ Computer Science, Jacobs University, Bremen (DE)
`initial.lastname@jacobs-university.de`

² Department of Computer Science, University of Bologna (IT)
`sacerdot@cs.unibo.it`

Abstract. The integration of reasoning and computation services across system and language boundaries has been mostly treated from an engineering perspective. In this paper we take a foundational point of view. We identify the following form of integration problems: an informal (mathematical; i.e, logically underspecified) specification has multiple concrete formal implementations between which queries and results have to be transported. The integration challenge consists in dealing with the implementation-specific details such as additional constants and properties. We pinpoint their role in safe and unsafe integration schemes and propose a proof-theoretic solution based on modular theory-graphs that include the meta-logical foundations. This also gives a clean conceptual basis for earlier attempts that explain integration via “content/semantic markup”.

1 Introduction

The aim of integrating Computer Algebra Systems (CAS) and Deduction Systems (DS) is twofold: to bring the efficiency of CAS algorithms to DS (without sacrificing correctness) and to bring the correctness assurance of the proof theoretic foundations of DS to CAS computations (without sacrificing efficiency). In general, the integration of computation and reasoning systems can be organized either by extending the internals of one systems by methods (data structure and algorithms) from the other, or by passing representations of mathematical objects and system state between independent systems, thus delegating parts of the computation to more efficient or secure platforms. We will deal with the latter approach here, which again has two distinct sets of problems. The first addresses engineering problems and revolves about communication protocol questions like shared state, distributed garbage collection, and translating input syntaxes of the different systems. The syntax questions have been studied extensively in the last decade and led to universal content markup languages for mathematics like MathML and OpenMath to organize communication. The second set of problems comes from the fact that passing mathematical objects between systems can only be successful if their meaning is preserved in the communication. This meaning is given via logical consequence in the logical system together with the axioms and definitions of (or inscribed in) the respective systems.

We will address this in the current paper, starting from the observation that content level communication between mathematical systems, to be effective, cannot always respect logical consequence. On the other hand, there is the problem of trusting the communication itself, that boils down to studying the preservation of logical consequence. Surprisingly, this problem has not received in the literature the attention it deserves. Moreover, the problem of faithful safe communication, which preserves not only the consequence relation but also the intuitive meaning of a formal object, is not even always perceived as a structural problem of content level languages.

For example, people with a strong background in first order logic tend to assume that faithful and safe communication can always be achieved simply by strengthening the specifications; others believe that encoding logical theories is already sufficient for safe communication and do not appreciate that the main problem is just moved to faithfulness. Several people from the interactive theorem proving world have raised concerns about trusting CAS and solved the issue by re-checking the results or the traces of the computation (here called proof sketches). Sometimes this happens under the assumption that the computation is already correct and just needs to be re-checked, neglecting the interesting case when the proof sketch cannot be refined to a valid proof (or computation) without major patching (see [DdC99] for a special case).

In section 3, we analyze the integration problem for mathematical systems from a formal position — basing our deliberations on the consequence relation. Our integration framework arises by adding some key innovations to the MMT language described in [RK10], which arises as a generalization of OpenMath and a clarification of OMDoc. Therefore, we sketch the MMT framework first in Sect. 2. Then we describe how integration can be realized our framework using partial theory morphisms in Sect. 4. Sect. 5 discusses related work and Sect. 6 concludes the paper.

2 The MMT Language

Agreeing on a common syntax like OpenMath is the first step towards system integration. This already enables a number of structural services such as storage and transport or editing and browsing that they do not depend on the semantics of the processed expressions. But while we have a good solution for a joint syntax, it is significantly harder to agree on a joint semantics. Fixing a semantics for a system requires a foundational commitment that excludes systems based on other foundations. The weakness of the (standard) OpenMath content dictionaries can be in part explained by this problem: The only agreeable content dictionaries are those where any axioms (formal or informal) are avoided that would exclude some foundations.

MMT was designed to overcome this problem by placing it in between frameworks like OpenMath and OMDoc on the one hand and logical frameworks like LF and CIC on the other hand. The basic idea is that a system's foundation itself is represented as a content dictionary. Thus, both meta and object lan-

guage are represented uniformly as MMT *theories*. Furthermore, *theory morphisms* are employed to translate between theories, which makes MMT expressive enough to represent translation between meta-languages and thus to support cross-foundation integration. As MMT permits the representation of logics as theories and internalizes the meta-relation between theories, this provides the starting point to analyze the cross-foundation integration challenge within a formal framework.

Syntax We will work with a very simple fragment of the MMT language that suffices for our purposes, and refer to [RK10] for the full account. It is given by the following grammar where $[-]$ denotes optional parts and T, v, c , and x are identifiers:

Theory graph	γ	$::=$	$\cdot \mid \gamma, T \stackrel{[T]}{=} \{\vartheta\} \mid \gamma, v : T \rightarrow T \stackrel{v}{=} \{\sigma\}$
Theory body	ϑ	$::=$	$\cdot \mid \vartheta, c[: O] [= O']$
Morphism body	σ	$::=$	$\cdot \mid \sigma, c \mapsto O$
Objects	O	$::=$	OpenMath objects
Morphisms	μ	$::=$	$v \mid id_T \mid \mu \circ \mu$
Contexts	C	$::=$	$x_1 : O_1, \dots, x_n : O_n$
Substitutions	s	$::=$	$x_1 := O_1, \dots, x_n := O_n$

In particular, we omit the module system of MMT that permits imports between theories.

$T \stackrel{L}{=} \{\vartheta\}$ declares a *theory* T with *meta-theory* L defined by the list ϑ of symbol declarations. The intuition of meta-theories is that L is the meta-language that declares the foundational symbols used to type and define the symbol declarations in ϑ .

All *symbol declarations* in a theory body are of the $c : O = O'$. This declares a new symbol c where both the type O and the definiens O' are optional. If given, they must be T -objects, which are defined as follows. A symbol is called *accessible* to T if it is declared in T or accessible to the meta-theory of T . An OpenMath object is called a *T -object* if it only uses symbols that are accessible to T .

Example 1. Consider the natural numbers defined within the calculus of constructions (see [BC04]). We represent this in MMT using a theory **CIC** declaring untyped, undefined symbols such as **Type**, λ and \rightarrow . Then **Nat** is defined as a theory with meta-theory **CIC** giving symbol declarations such as $\mathbb{N} : \text{OMS}(\text{cd} = \text{CIC}, \text{name} = \text{Type})$ or $\text{succ} : \text{OMA}(\text{OMS}(\text{cd} = \text{CIC}, \text{name} = \rightarrow), \text{OMS}(\text{cd} = \text{Nat}, \text{name} = \mathbb{N}), \text{OMS}(\text{cd} = \text{Nat}, \text{name} = \mathbb{N}))$.

S-contexts C are lists of variable declarations $\dots, x_i : O_i, \dots$ for S -objects O_i . *S-substitutions* s for an S -context C are lists of variable assignments $\dots, x_i := o_i, \dots$. In an object O in context C , exactly the variables in C may occur freely; then for a substitution s for C , we write $O[s]$ for the result of replacing every free occurrence of x_i with o_i .

Relations between MMT theories are expressed using theory morphisms. Given two theories S and T , a *theory morphism* from S to T is declared using $v : S \rightarrow T \stackrel{L}{=} \{\sigma\}$. Here σ must contain one assignment $c \mapsto O$ for every

symbol c declared in the body of S , and for some T -objects O . If S and T have meta-theories L and M , then v must also include a meta-morphism $l : L \rightarrow M$.

Every $v : S \rightarrow T \stackrel{L}{=} \{\sigma\}$ induces a *homomorphic extension* $v(-)$ that maps S -objects to T -objects. $v(-)$ is defined by induction on the structure of OpenMath objects. The base case $v(c)$ for a symbol c is defined as follows: If c is accessible to the meta-theory of S , we put $v(c) := l(c)$; otherwise, we must have $c \mapsto O$ in σ , and we put $v(c) := O$. $v(-)$ also extends to contexts and substitutions in the obvious way.

By experimental evidence, all declarative languages for mathematics currently known can be represented naturally in MMT. In particular, MMT uses the Curry-Howard representation [CF58,How80] of propositions as types and proofs as terms. Thus, an axiom named a asserting F is a special case of a symbol a of type F , and a theorem named t asserting F with proof p is a special case of a symbol t with type F and defines p . All inference rules needed to form p , are symbols declared in the meta-theory.

Semantics The use of meta-theories makes the logical foundation of a system part of an MMT theory and makes the syntax of MMT foundation-independent. The analogue for the semantics is more difficult to achieve: The central idea is that the semantics of MMT is parametric in the semantics of the foundation.

To make this precise, we call a theory without a meta-theory *foundational*. A *foundation* for MMT consists of a foundational theory L and two judgments for typing and equality of objects:

- $\gamma; C \vdash_T O : O'$ states that O is a T -object over C typed by the T -object O' ,
- $\gamma; C \vdash_T O = O'$ states the equality of two T -objects over C ,

defined for an arbitrary theory T declared in γ with meta-theory L . In particular, MMT does not distinguish terms, types, and values at higher universes — all expressions are OpenMath objects with an arbitrary binary typing relation between them. We will omit C when it is empty.

These judgments are similar to those used in almost all declarative languages, except that we do not commit to a particular inference system — all rules are provided by the foundation and are transparent to MMT except for the rules for the base cases of T -objects:

$$\frac{T \stackrel{L}{=} \{\vartheta\} \text{ in } \gamma \quad c : O = O' \text{ in } \vartheta}{\gamma \vdash_T c : O} \mathcal{T}_= \quad \frac{T \stackrel{L}{=} \{\vartheta\} \text{ in } \gamma \quad c : O = O' \text{ in } \vartheta}{\gamma \vdash_T c = O'} \mathcal{T}_=$$

and accordingly if O or O' are omitted. For example, adding the usual rules for the calculus of constructions yields a foundation for the foundational theory CIC.

- Given a foundation, MMT defines (among others) the two judgments
- $\gamma \vdash \mu : S \rightarrow T$ states that μ is a theory morphism from S to T ,
 - if $\gamma \vdash \mu_i : S \rightarrow T$, then $\gamma \vdash \mu_1 = \mu_2$ states that $\vdash_T \mu_1(c) = \mu_2(c)$ for all symbols c that are accessible to S ,
 - $\gamma \vdash_S s : C$ states that s is a well-typed for C , i.e., for every $x_i := o_i$ in s and $x_i : O_i$ in C , we have $\gamma \vdash_S o_i : O_i$,

– $\gamma \vdash \mathcal{G}$ states that \mathcal{G} is a well-formed theory graph.

In the sequel, we will omit γ if it is clear from the context.

The most important MMT rule for our purposes is the rule that permits adding an assignment to a theory morphism: If S contains a declaration $c : O_1 = O_2$, then a theory morphism $v : S \rightarrow T \stackrel{l}{=} \{\sigma\}$ may contain an assignment $c \mapsto O$ only if $\vdash_T O : v(O_1)$ and $\vdash_T O = v(O_2)$. The according rule applies if c has no type or no definiens. Of course, this means that assignments $c \mapsto O$ are redundant if c has a definiens; but it is helpful to state the rule in this way to prepare for our definitions below.

Due to these rules, we obtain that if $\gamma \vdash \mu : S \rightarrow T$ and $\vdash_S O : O'$ or $\vdash_S O = O'$, then $\vdash_T \mu(O) : \mu(O')$ and $\vdash_T \mu(O) = \mu(O')$, respectively. Thus, typing and equality are preserved along theory morphisms.

Due to the Curry-Howard representation, this includes the preservation of provability: $\vdash_T p : F$ states that p is a well-formed proof of F in T . And if S contains an axiom $a : F$, a morphism μ from S to T must map a to a T -object of type $\mu(F)$, i.e., to a T -proof of $\mu(F)$. This yields the well-known intuition of a theory morphism. In particular, if μ is the identity on those symbols that do not represent axioms, then $\vdash \mu : S \rightarrow T$ implies that every S -theorem is an T -theorem.

3 Integration Challenges

In this section, we will develop some general intuitions about system integration and then give precise definitions in MMT. A particular strength of MMT is that we can give these precise definition without committing to a particular foundational system and thus without loss of generality.

The typical integration situation is that we have two systems \mathcal{S}_i for $i = 1, 2$ that implement a shared specification *Spec*. For example, these systems can be computer algebra systems or (semi-)automated theorem provers. Our integration goal is to move problems and results between \mathcal{S}_1 and \mathcal{S}_2 .

Specifications and Systems Let us first assume a single system \mathcal{S} implementing *Spec*, whose properties are given by logical consequence relations \Vdash_{Spec} and $\Vdash_{\mathcal{S}}$. We call \mathcal{S} *sound* if $\Vdash_{\mathcal{S}} F$ implies $\Vdash_{Spec} F$ for every formula F in the language of *Spec*. Conversely, we call \mathcal{S} *complete* if $\Vdash_{Spec} F$ implies $\Vdash_{\mathcal{S}} F$.

While these requirements seem quite natural at first, they are too strict for practical purposes. It is well-known that soundness fails for many CASs, which compute wrong results by not checking side conditions during simplification. Reasons for incompleteness can be theoretical — e.g., when \mathcal{S} is a first-order prover and *Spec* a higher-order specification — or practical — e.g., due to resource limitations.

Moreover, soundness also fails in the case of underspecification: \mathcal{S} is usually much stronger than *Spec* because it must commit to concrete definitions and implementations for operations that are loosely specified in *Spec*. A typical example is the representation of undefined terms (see [Far04] for a survey of techniques).

If $Spec$ specifies the rational numbers using in particular $\forall x. x \neq 0 \Rightarrow x/x = 1$, and \mathcal{S} defines $1/0 = 2/0 = 0$, then \mathcal{S} is not sound because $1/0 = 2/0$ is not a theorem of $Spec$.

We can define the above notions in MMT as follows. A *specification* $Spec$ is an MMT theory; its meta-theory (if any) is called the *specification language*. A system implementing $Spec$ consists of an MMT theory \mathcal{S} and an MMT theory morphism $v : Spec \rightarrow \mathcal{S}$; the meta-theory of \mathcal{S} (if any) is called the *implementation language*. With this definition and using the Curry-Howard representation of MMT, we can provide a deductive system for the consequence relations used above: $\Vdash_{Spec} F$ iff there is a p such that $\vdash_{Spec} p : F$; and accordingly for $\Vdash_{\mathcal{S}}$.

In the simplest case, the morphism v is an inclusion, i.e., for every symbol in $Spec$, \mathcal{S} contains a symbol of the same name. Using an arbitrary morphism v provides more flexibility, for example, the theory of the natural numbers with addition and multiplication implements the specification of monoids in two different ways via two different morphisms.

Example 2. We use a theory for second-order logic as the specification language; it declares symbols for $\forall, =$, etc. $Spec = Nat$ is a theory for the natural numbers; it declares symbols $N, 0$ and $succ$ as well as one symbol $a : F$ for each Peano axiom F .

For the implementation language, we use a theory **ZF** for ZF set theory; it has meta-theory first-order logic and declares symbols for **set**, \in , \emptyset , etc. Then we can implement the natural numbers in a theory $\mathcal{S} = \mathbf{Nat}$ declaring, e.g., a symbol $\mathbf{0}$ defined as \emptyset , a symbol **succ** defined such that $\mathbf{succ}(n) = n \cup \{n\}$, and prove one theorem $a : F = p$ in \mathcal{S} for each Peano axiom. Note that **Nat** yields theorems about the natural numbers that cannot be expressed in $Spec$, for example $\Vdash_{\mathbf{ZF}} 0 \in 1$. We obtain a morphism $\mu_1 : Nat \rightarrow \mathbf{Nat}$ using $N \mapsto \mathbf{N}$, $0 \mapsto \mathbf{0}$ etc.

Continuing Ex. 1, we obtain a different implementation $\mu_2 : Nat \rightarrow \mathbf{Nat}$ using $N \mapsto \mathbf{N}$, $0 \mapsto \mathbf{0}$ etc.

To capture practice in formal mathematics, we have to distinguish between the definitional and the axiomatic method. The *axiomatic* method fixes a formal system L and then describes mathematical notions in L -theories T using free symbols and axioms. T is interpreted in models, which may or may not exist. This is common in model theoretical logics, especially first-order logic, and in algebraic specification. In MMT, T is represented as a theory with meta-theory L and with only undefined constants. In Ex. 2, L is second-order logic and T is $Spec$.

The *definitional* method, on the other hand, fixes a formal system L together with a minimal theory T_0 and then describes mathematical notions using definitional extensions T of T_0 . The properties of the notions defined in T_0 are derived as theorems. The interpretation of T is uniquely determined given a model of T_0 . This is common in proof theoretical logics, especially LCF-style proof assistants, and in set theory. In Ex. 2, L is first-order logic, T_0 is ZF, and T is \mathcal{S} .

Types of Integration Let us now consider a specification $Spec$ and two implementations $\mu_i : Spec \rightarrow \mathcal{S}_i$. To simplify the notation, we will write \vdash and \vdash_i instead of \vdash_{Spec} and $\vdash_{\mathcal{S}_i}$. We first describe different ways how to integrate \mathcal{S}_1 and \mathcal{S}_2 intuitively.

Borrowing means to use \mathcal{S}_1 to prove theorems in the language of \mathcal{S}_2 . Thus, the input to \mathcal{S}_1 is a conjecture F and the output is an expression $\vdash_1 p : F$. In general, since MMT does not prescribe a calculus for proofs, the object p can be a formal proof term, a certificate, proof sketch, or simply a yes/no answer.

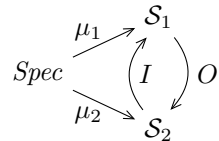
Computation means to reuse a \mathcal{S}_1 computation in \mathcal{S}_2 . Thus, the input of \mathcal{S}_1 is an expression t , and the output is a proof p with an expression t' such that $\vdash_1 p : t = t'$. To be useful, t' should be simpler than t in some way, e.g., maximally simplified or even normalized.

Querying means answering a query in \mathcal{S}_1 and transferring the results to \mathcal{S}_2 . This is similar to borrowing in that the input to \mathcal{S}_1 is a formula F . However, now F may contain free variables, and the output is not only a proof p but also a substitution s for the free variables such that $\vdash_1 p : F[s]$.

In all cases, a translation I must be employed to translate the input from \mathcal{S}_1 to \mathcal{S}_2 . Similarly, we need a translation O in the opposite direction to translate the output t' and s and (if available) p from \mathcal{S}_2 to \mathcal{S}_1 .

To define these integration types formally in MMT, we first note that borrowing is a special case of querying if F has no free variables. Similarly, computation is a special case of querying if F has the form $t = X$ for a variable X that does not occur in t .

To define querying in MMT, we assume a specification, two implementations, and morphisms I and O as on the right. I and O must satisfy $O \circ I = id_{\mathcal{S}_2}$, $O \circ \mu_1 = \mu_2$, and $I \circ \mu_2 = \mu_1$. Then we obtain the following *general form of an integration problem*: Given an \mathcal{S}_2 -context C and a query



$C \vdash_2 ? : F$ (where $?$ denotes the requested proof), find a substitution $\vdash_1 s : I(C)$ and a proof $\vdash_1 p : I(F)[s]$. Then MMT guarantees that $\vdash_2 O(p) : F[O(s)]$ so that we obtain $O(s)$ as the solution. Moreover, only the existence of O is necessary but not O itself — once a proof p is found in \mathcal{S}_1 , the existence of O ensures that F is true in \mathcal{S}_2 , and it is not necessary to translate p to \mathcal{S}_2 .

We call the above scenario *safe bidirectional communication* between \mathcal{S}_1 and \mathcal{S}_2 because I and O are theory morphisms and thus guarantee that consequence and truth are preserved in both directions. This scenario is often implicitly assumed by people coming from the first-order logic community. Indeed, if \mathcal{S}_1 and \mathcal{S}_2 are automatic or interactive theorem provers for first-order logic, then the logic of the two systems is the same and both \mathcal{S}_1 and \mathcal{S}_2 are equal to $Spec$.

If we are only interested in *safe directed communication*, i.e., transferring results from \mathcal{S}_1 to \mathcal{S}_2 , then it is sufficient to require only O . Indeed, often μ_2 is an inclusion, and the input parameters C and F , which are technically \mathcal{S}_2 -objects, only use symbols from $Spec$. Thus, they can be moved directly to $Spec$ and \mathcal{S}_1 , and I is not needed.

Similarly, the substitution s can often be stated in terms of *Spec*. In that case, O is only needed to translate the proof p . If the proof translation is not feasible, O may be omitted as well. Then we speak of *unsafe communication* because we do not have a guarantee that the communication of results is correct. For example, let \mathcal{S}_1 and \mathcal{S}_2 be two CASSs, that may compute wrong results by not checking side conditions during simplification. Giving a theory morphism O means that the “bugs” of the system \mathcal{S}_1 must be “compatible” with the “bugs” of \mathcal{S}_2 , which is quite unlikely.

The above framework for safe communication via theory morphisms is particularly appropriate for the integration of axiomatic systems. However, if \mathcal{S}_1 and \mathcal{S}_2 employ different mathematical foundations or different variants of the same foundation, it can be difficult to establish the necessary theory morphisms. In MMT, this means that \mathcal{S}_1 and \mathcal{S}_2 have different meta-theories so that I and O must include a meta-morphism. Therefore, unsafe communication is often used in practice, and even that can be difficult to implement.

Our framework is less appropriate if \mathcal{S}_1 or \mathcal{S}_2 are developed using the definitional method. For example, consider Aczel’s encoding of set theory in type theory [Acz99, Wer97]. Here $\mathcal{S}_1 = \mathbf{Nat}$ as in Ex. 2, and $\mathcal{S}_2 = \mathbf{Nat}$ as in Ex. 1. Aczel’s encoding provides the needed meta-morphism $l : \mathbf{ZF} \rightarrow \mathbf{CIC}$ of O . But because \mathbf{Nat} is definitional, we already have $O = l$, and we have no freedom to define O such that it maps the concepts of \mathbf{Nat} to their counterparts in \mathbf{Nat} . Formally, in MMT, this means that the condition $O \circ \mu_1 = \mu_2$ fails. Instead, we obtain two versions of the natural numbers in \mathbf{CIC} : a native one given by μ_2 and the translation of \mathbf{Nat} given by $O \circ \mu_1$. Indeed, the latter must satisfy all \mathbf{ZF} -theorems including, e.g., $0 \in 1$, which is not even a well-formed formula over \mathbf{Nat} . We speak of *faithful communication* if $O \circ \mu_1 = \mu_2$ can be established even when \mathcal{S}_1 is definitional. This is not possible in MMT without the extension we propose below.

4 A Framework for System Integration

In order to realize faithful communication within MMT, we introduce *partial theory morphisms* that can filter out those definitional details of \mathcal{S}_1 that need not and cannot be mapped to \mathcal{S}_2 . We will develop this new concept in general in Sect. 4.1 and then apply it to the integration problem in Sect. 4.2.

4.1 Partial Theory Morphisms in MMT

Syntax We extend the MMT syntax with the production $O ::= \top$. The intended use of \top is to put assignments $c \mapsto \top$ into the body of a morphism $v : S \rightarrow T \stackrel{l}{=} \{\sigma\}$ in order to make v undefined at c . We say that v *filters* c . The homomorphic extension $v(-)$ remains unchanged and is still total: If O contains filtered symbols, then $v(O)$ contains \top as a subobject. In that case, we say v *filters* O .

Semantics We refine the semantics as follows. A *dependency cut* D for an MMT theory T is a pair (D_{type}, D_{def}) of two sets of symbols accessible to T . Given such a dependency cut, we define *dependency-aware judgments* $\gamma \vdash_D O : O'$ and $\gamma \vdash_D O = O'$ as follows. $\gamma \vdash_D O : O'$ means that there is a derivation of $\gamma \vdash_T O : O'$ that uses the rules \mathcal{T}_\neq and $\mathcal{T}_=$ at most for the constants in D_{type} and D_{def} , respectively. $\gamma \vdash_D O = O'$ is defined accordingly.

In other words, if we have $\gamma' \vdash_D O : O'$ and obtain γ' by changing the type of any constant not in D_{type} or the definiens of any constant not in D_{def} , then we still have $\gamma' \vdash_D O : O'$. Then a *foundation* consists of a foundational theory L together with dependency-aware judgments for typing and equality whenever T has meta-theory L .

We make a crucial change to the MMT rule for assignments in a theory morphism: If S contains a declaration $c : O_1 = O_2$, then a theory morphism $v : S \rightarrow T \stackrel{l}{=} \{\sigma\}$ may contain the assignment $c \mapsto O$ only if the following two conditions hold: (i) if O_1 is not filtered by v , then $\vdash_T O : v(O_1)$; (ii) if O_2 is not filtered by v , then $\vdash_T O = v(O_2)$. The according rule applies if O_1 or O_2 are omitted.

In [RK10], a stricter condition is used. There, if O_1 or O_2 are filtered, then c must be filtered as well. While this is a natural strictness condition for filtering, it is inappropriate for our use cases: For example, filtering all L -symbols would entail filtering all S -symbols.

Our weakened strictness condition is still strong enough to prove the central property of theory morphisms: If $\gamma \vdash \mu : S \rightarrow T$ and $\vdash_D O : O'$ for some $D = (D_{type}, D_{def})$ and v does not filter O, O' , the type of a constant in D_{type} , or the definiens of a constant in D_{def} , then $\vdash_T \mu(O) : \mu(O')$. The according result holds for the equality judgment.

Finally, we define the *weak equality of morphisms* $\mu_i : S \rightarrow T$. We define $\vdash \mu_1 \leq \mu_2$ in the same way as $\vdash \mu_1 = \mu_2$ except that $\vdash_T \mu_1(c) = \mu_2(c)$ is only required if c is not filtered by μ_1 . We say that $\vdash \eta : T \rightarrow S$ is a *partial inverse* of $\mu : S \rightarrow T$ if $\vdash \eta \circ \mu = id_S$ and $\vdash \mu \circ \eta \leq id_T$.

Example 3. Consider the morphism $\mu_1 : Nat \rightarrow \mathbf{Nat}$ from Ex. 2. We build its partial inverse $\eta : \mathbf{Nat} \rightarrow Nat \stackrel{l}{=} \{\sigma\}$. The meta-morphism l filters all symbols of \mathbf{ZF} , e.g., $l(\emptyset) = \top$. Then the symbol \mathbf{N} of \mathbf{Nat} has filtered type and filtered definiens. Therefore, the conditions (i) and (ii) above are vacuous, and we use $\mathbf{N} \mapsto N$ in σ . Then all remaining symbols of \mathbf{Nat} (including the theorems) have filtered definiens but unfiltered types. For example, for $\mathbf{0} : \mathbf{N} = \emptyset$ we have $\eta(\emptyset) = \top$ but $\eta(\mathbf{N}) = N$. Therefore, condition (ii) is vacuous, and we map these symbols to their counterparts in Nat , e.g., using $\mathbf{0} \mapsto 0$ in σ . These assignments are type-preserving as required by condition (i) above, e.g., $\vdash_{Nat} \eta(\mathbf{0}) : \eta(\mathbf{N})$.

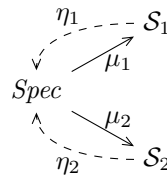
4.2 Integration via Partial Theory Morphisms

The following gives a typical application of our framework by safely and faithfully communicating proofs from a stronger to a weaker system:

Example 4. In [IR11], we gave formalizations of Zermelo-Fraenkel (*ZFC*) set theory and Mizar’s Tarski-Grothendieck set theory (*TG*) using the logical framework *LF* as the common meta-theory. *ZFC* and *TG* share the language of first-order set theory. But *TG* is stronger than *ZFC* because of Tarski’s axiom, which implies, e.g, the sentence *I* stating the existence of infinite sets (which is an axiom in *ZFC*) and large cardinals (which is unprovable in *ZFC*). For example, we have an axiom $a_\infty : I$ in *ZFC*, and an axiom $tarski : T$ and a theorem $t_\infty : I = P$ in *TG*. Many *TG*-theorems do not actually depend on this additional strength, but they do depend on t_∞ and thus indirectly on *tarski*.

Using our framework, we can capture such a theorem as the case of a *TG*-theorem $\vdash_D p : F$ where *F* is the theorem statement and $t_\infty \in D_{type}$ but $t_\infty \notin D_{def}$ and $tarski \notin D_{type}$. We can give a partial theory morphism $v : TG \rightarrow ZFC \stackrel{id_{LF}}{=} \{\dots, t_\infty \mapsto a_\infty, \dots\}$. Then *v* does not filter *p*, and we obtain $\vdash_{ZFC} v(p) : F$.

Assume now that we have two implementations $\mu_i : Spec \rightarrow \mathcal{S}_i$ of *Spec* and partial inverses η_i of μ_i , where \mathcal{S}_i has meta-theory L_i . This leads to the diagram on the right where (dashed) edges are (partial) theory morphisms. We can now obtain the translations $I : \mathcal{S}_2 \rightarrow \mathcal{S}_1$ and $O : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ as $I = \mu_1 \circ \eta_2$ and $O = \mu_2 \circ \eta_1$. Note that *I* and *O* are partial inverses of each other.



As in Sect. 3, let $C \vdash_2? : F$ be a query in \mathcal{S}_2 . If η_2 does not filter any symbols in *C* or *F*, we obtain the translated problem $I(C) \vdash_1? : I(F)$. Let us further assume that there is an \mathcal{S}_1 -substitution $\vdash_1 s : I(C)$ and a proof $\vdash_1 p : I(F)[s]$ such that *p* and *s* are not filtered by η_1 . Because *I* and *O* are mutually inverse and morphism application preserves typing, we obtain the solution $\vdash_2 O(p) : F[O(s)]$.

The condition that η_2 does not filter *C* and *F* is quite reasonable in practice: Otherwise, the meaning of the query would depend on implementation-specific details of \mathcal{S}_2 , and it is unlikely that \mathcal{S}_1 should be able to find an answer anyway. On the other hand, the morphism η_1 is more likely to filter the proof *p*. Moreover, since the proof must be translated from L_1 to L_2 passing through *Spec*, the latter must include a proof system to allow translation of proofs. In practice this is rarely the case, even if the consequence relation of *Spec* can be expressed as an inference system. For example, large parts of mathematics or the OpenMath content dictionaries implicitly (import) first-order logic and ZF set theory.

We outline two ways how to remedy this: We can *communicate filtered proofs* or change the morphisms to *widen the filters* to let more proofs pass.

Communicating Filtered Proofs Firstly, if the proof rules of \mathcal{S}_1 are filtered by η_1 , what is received by \mathcal{S}_2 after applying the output translation *O* is a filtered proof, i.e., a proof object that contains the constant \top . \top represents gaps in the proof that were lost in the translation.

In an extreme case, all applications of proof rules become \top , and the only unfiltered parts of $O(p)$ are formulas that occurred as intermediate results during the proof. In that case, $O(p)$ is essentially a list of formulas F_i (a proof sketch

in the sense of [Wie03]) such that $I(F_1) \wedge \dots \wedge I(F_{i-1}) \vdash_1 I(F_i)$ for $i = 1, \dots, n$. In order to refine $O(p)$ into a proof, we have to derive $\vdash_1 F_n$. Most of the time, it will be the case that $F_1, \dots, F_{i-1} \vdash_2 F_i$ for all i , and the proof is obtained compositionally if \mathcal{S}_2 can fill the gaps through automated reasoning. When this happens, the proof sketch is already a complete declarative proof.

Example 5. Let \mathcal{S}_1 and \mathcal{S}_2 be implementations of the rational numbers with different choices for division by zero. In \mathcal{S}_1 , division by zero yields a special value for undefined results, and operations on undefined values yield undefined results; then we have the \mathcal{S}_1 -theorem t asserting $\forall a, b, c. a(b/c) \doteq (ab)/c$. In \mathcal{S}_2 , we have $n/0 \doteq 1$ and $n\%0 \doteq n$; then we have the \mathcal{S}_2 -theorems t_1, t_2, t_3 asserting $\forall m, n. n \doteq (n/m) * m + n\%m$, $\forall m. m/m \doteq 1$, and $\forall m. m\%m \doteq 0$.

The choice in \mathcal{S}_2 reduces the number of case analyses in basic proofs. But t is not a theorem of \mathcal{S}_2 ; instead, we only have a theorem t' asserting $\forall a, b, c. c \neq 0 \Rightarrow a(b/c) \doteq (ab)/c$. On the other hand, \mathcal{S}_1 is closer to common mathematics, but the t_i are not theorems of \mathcal{S}_1 because the side condition $m \neq 0$ is needed.

Hence, we do not have a total theory morphism $O : \mathcal{S}_1 \rightarrow \mathcal{S}_2$, but we can give a partial theory morphism O that filters t . Now consider, for example, a proof p over \mathcal{S}_1 that instantiates t with some values A, B, C . When translating p to \mathcal{S}_2 , t is filtered, but we can still communicate p , and \mathcal{S}_2 can treat $O(p)$ as a proof sketch. Typically, t is applied in a context where $C \neq 0$ is known anyway so that \mathcal{S}_2 can patch $O(p)$ by using t' — which can easily be found by automated reasoning.

Integration in the other direction works accordingly.

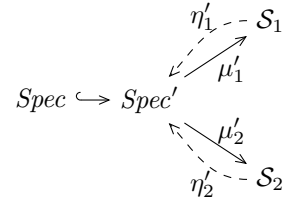
Widening the Filters An alternative solution is to use additional knowledge about \mathcal{S}_1 and \mathcal{S}_2 to obtain a translation where $O(p)$ is not filtered. In particular, if p is filtered completely, we can strengthen $Spec$ by adding an inference system for the consequence relation of $Spec$, thus obtaining $Spec'$. Then we can extend the morphisms μ_i accordingly to μ'_i , which amounts to proving that \mathcal{S}_i is a correct implementation of $Spec$.

Now η_i can be extended as well so that its domain becomes bigger, i.e., the morphism η_1 and thus O filter less proofs and become “wider”.

Note that we are flexible in defining $Spec'$ as required by the particular choices of L_1 and L_2 . That way the official specification remains unchanged, and we can maximize the filters for every individual integration scenario.

Example 6 (Continuing Ex. 3). A typical situation is that we have a theorem F over \mathbf{Nat} whose proof p uses the Peano axioms and the rules of first-order logic but does not expand the definitions of the natural numbers. Moreover, if $a : A = P$ is a theorem in \mathbf{Nat} that establishes one of the Peano axioms, then p will refer to a , but will not expand the definition of a . Formally, we can describe this as $\vdash_D p : F$ where $\mathbf{0}, a \in D_{type}$ but $\mathbf{0}, a \notin D_{def}$.

We can form $Spec'$ by extending $Spec$ with proof rules for first-order logic and extend η to η' accordingly. Since η does not filter the types of $\mathbf{0}$ and a , we



obtain a proof $\vdash_{Spec} \eta'(p) : \eta'(F)$ due to the type-preservation properties of our partial theory morphisms. Despite the partiality of η' , the correctness of this proof is guaranteed by the framework.

Both ways to integrate systems are not new and have been used ad hoc in concrete integration approaches, see Sect. 5. With our framework, we are able to capture them in a rigorous framework where their soundness can be studied formally.

5 Related Work

In the MoWGLI project [MoW], semantics markup is introduced to distinguish from specifications given in the content markup languages OpenMath and OM-Doc, and the implementations given in the calculus of constructions. This corresponds closely to the use of meta-theories in MMT: Their content markup corresponds to MMT theories without meta-theory; and semantics markup corresponds to MMT theories with meta-theory CIC.

A framework very similar to ours was given in [CFW03]. Our MMT theories with meta-theory correspond to their biform theories, except that the latter adds algorithms. Our theory morphisms I and O correspond to their translations `export` and `import`.

Integration by borrowing is the typical scenario of integrating theorem provers and proof assistants. For example, Leo-II [BPTF07] or the Sledgehammer tactic of Isabelle [Pau94] (\mathcal{S}_2) use first-order provers (\mathcal{S}_1) to reason in higher-order logic. Here the input translation I is partial inverse of the inclusion from first-order logic to higher-order logic. A total translation from modal logic to first-order logic is used in [HS00]. In all cases, the safety is verified informally on the meta-level and no output translation O in our sense is used. But Isabelle makes the communication safe by reconstructing a proof from the proof (sketch) returned by the prover.

The above systems are called on demand using an input translation I . Alternatively a collection of \mathcal{S}_1 -proofs can be translated via an output translation O for later reuse in \mathcal{S}_2 ; in that case no input translation I is used at all. Examples are the translations from Isabelle/HOL in HOL Light [McL06], from HOL Light to Isabelle/HOL [OS06], from HOL Light to Coq [KW10], or from Isabelle/HOL to Isabelle/ZF [KS10].

The translation from HOL to Isabelle/HOL is notable because it permits faithful translations, e.g., the real numbers of HOL can be translated to the real numbers of Isabelle/HOL, even though the two systems define them differently. The safety of the translation is achieved by recording individual \mathcal{S}_1 -proofs and replaying them in \mathcal{S}_2 . This was difficult to achieve even though \mathcal{S}_1 and \mathcal{S}_2 are based on the same logic.

The translation given in [KW10] is the first faithful translation from HOL proofs to CIC proofs. Since the two logics are different, in order to obtain a total map the authors widen the filter by assuming additional axioms on CIC (excluded middle and extensionality of functions). This technique is not exploitable

when the required axioms are inconsistent. Moreover, the translation is suboptimal, since it uses excluded middle also for proofs that are intuitionistic. To improve the solution, we could use partial theory morphisms that map case analysis over boolean in HOL to \top , and then use automation to avoid excluded middle in CIC when the properties involved are all decidable.

In all above examples but [KW10], the used translations are not verified within a logical framework. The Logosphere [PSK⁺03] project used the proof theoretical framework LF to provide statically verified logic translations that permit inherently safe communication. The most advanced such proof translation is one from HOL to Nuprl [NSM01].

The theory of institutions [GB92] provides a general model theoretical framework in which borrowing has been studied extensively [CM97] and implemented successfully [MML07]. Here the focus is on giving the morphism I explicitly and using a model theoretical argument to establish the existence of some O ; then communication is safe without explicitly translating proofs.

Integration by computation is the typical scenario for the integration of computer algebra systems, which is the main topic of the Calculemus series of conferences. For typical examples, see [DM05] where the computation is performed by a CAS, and [AT07] where the computation is done by a term rewriting system. Communication is typically unsafe. Alternatively, safety can be achieved if the results of the CAS — e.g., the factorization of a polynomial — can be verified formally in a DS as done in [HT98] and [Sor00].

Typical applications of integration by querying are conjunctive query answering for a description logic. For example, in [TSP08], a first-order theorem prover is used to answer queries about the SUMO ontology.

The communication of filtered proofs essentially leads to formal proof sketch in the sense of [Wie03]. The idea of abstracting from a proof to a proof sketch corresponds to the assertion level proofs used in [Mei00] to integrate first-order provers. The recording and replaying of proof steps in [OS06] and the reconstruction of proofs in Isabelle are also special cases of the communication of filtered proofs.

6 Conclusion

In this paper we addressed problems of preserving the semantics in protocol-based integration of mathematical reasoning and computation systems. We analyzed the problem from a foundational point of view and proposed a framework based on theory graphs, partial theory morphisms, and explicit representations of meta-logics that allows to state solutions to the integration problem.

The main contribution and novelty of the paper is that it paves the way towards a *theory of integration*. We believe that this theory of integration is practical because it requires only a simple extension of the MMT framework, which already takes scalability issues very seriously [KRZ10].

References

- Acz99. Peter Aczel. On relating type theories and set theories. *Lecture Notes in Computer Science*, 1657, 1999.
- AT07. Andrea Asperti and Enrico Tassi. Higher order proof reconstruction from paramodulation-based refutations: The unit equality case. In *Calculus/MKM*, pages 146–160, 2007.
- BC04. Y. Bertot and P. Castéran. *Coq’Art: The Calculus of Inductive Constructions*. Springer, 2004.
- BPTF07. C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. The LEO-II Project. In *Automated Reasoning Workshop*, 2007.
- CF58. H. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.
- CFW03. J. Carette, W. Farmer, and J. Wajs. Trustable Communication between Mathematics Systems. In T. Hardin and R. Rioboo, editors, *Proceedings of Calculus*, pages 58–68, 2003.
- CM97. M. Cerioli and J. Meseguer. May I Borrow Your Logic? (Transporting Logical Structures along Maps). *Theoretical Computer Science*, 173:311–347, 1997.
- DdC99. David Delahaye and Roberto di Cosmo. Information retrieval in a Coq proof library using type isomorphisms. In *Proceedings of TYPES 99*, volume Lecture Notes in Computer Science. Springer-Verlag, 1999.
- DM05. David Delahaye and Micaela Mayero. Dealing with Algebraic Expressions over a Field in Coq using Maple. *Journal of Symbolic Computation (JSC)*, 39(5):569–592, May 2005.
- Far04. W. Farmer. Formalizing Undefinedness Arising in Calculus. In *IJCAR*, volume 3097 of *LNCS*, pages 475–489, 2004.
- GB92. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.
- How80. 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.
- HS00. U. Hustadt and R. Schmidt. MSPASS: Modal Reasoning by Translation and First-Order Resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, pages 67–71, 2000.
- HT98. J. Harrison and L. Théry. A Skeptic’s Approach to Combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.
- IR11. M. Iancu and F. Rabe. Formalizing Foundations of Mathematics. *Mathematical Structures in Computer Science*, 2011. To appear, see http://kwarc.info/frabe/Research/IR_foundations_10.pdf.
- KRZ10. M. Kohlhase, F. Rabe, and V. Zholudev. Towards MKM in the Large: Modular Representation and Scalable Software Architecture. In S. Autexier, J. Calmet, D. Delahaye, P. Ion, L. Rideau, R. Rioboo, and A. Sexton, editors, *Intelligent Computer Mathematics*, volume 6167 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2010.
- KS10. A. Krauss and A. Schropp. A Mechanized Translation from Higher-Order Logic to Set Theory. In M. Kaufmann and L. Paulson, editors, *Proceedings of the Interactive Theorem Proving conference*, 2010. to appear in *LNCS*.

- KW10. C. Keller and B. Werner. Importing HOL Light into Coq. In M. Kaufmann and L. Paulson, editors, *Proceedings of the Interactive Theorem Proving conference*, 2010. to appear in LNCS.
- McL06. S. McLaughlin. An Interpretation of Isabelle/HOL in HOL Light. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.
- Mei00. A. Meier. System Description: TRAMP: Transformation of Machine-Found Proofs into ND-Proofs at the Assertion Level. In D. McAllester, editor, *Automated Deduction*, volume 1831 of *Lecture Notes in Computer Science*, pages 460–464. Springer, 2000.
- MML07. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.
- MoW. MoWGLI project deliverables. http://mowgli.cs.unibo.it/html_no_frames/deliverables/index.html.
- NSM01. P. Naumov, M. Stehr, and J. Meseguer. The HOL/NuPRL proof translator - a practical approach to formal interoperability. In *14th International Conference on Theorem Proving in Higher Order Logics*. Springer, 2001.
- OS06. S. Obua and S. Skalberg. Importing HOL into Isabelle/HOL. In N. Shankar and U. Furbach, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.
- Pau94. L. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.
- PSK⁺03. F. Pfenning, C. Schürmann, M. Kohlhase, N. Shankar, and S. Owre. The Logosphere Project, 2003. <http://www.logosphere.org/>.
- RK10. F. Rabe and M. Kohlhase. A Scalable Module System. To be submitted, see <http://kwarc.info/frabe/Research/mmt.pdf>, 2010.
- Sor00. V. Sorge. Non-trivial Symbolic Computations in Proof Planning. In *FroCoS*, pages 121–135. Springer, 2000.
- TSP08. S. Trac, G. Sutcliffe, and A. Pease. Integration of the TPTPWorld into SigmaKEE. In B. Konev, R. Schmidt, and S. Schulz, editors, *Practical Aspects of Automated Reasoning*, volume 373 of *CEUR Workshop Proceedings*, 2008.
- Wer97. Benjamin Werner. Sets in types, types in sets. In Martin Abadi and Takahashi Ito editors, editors, *Theoretical Aspect of Computer Software TACS'97*, *Lecture Notes in Computer Science*, volume 1281, pages 530–546. Springer-Verlag, 1997.
- Wie03. Freek Wiedijk. Formal proof sketches. In Wan Fokkink and Jaco van de Pol, editors, *7th Dutch Proof Tools Day, Program + Proceedings*, 2003. CWI, Amsterdam.