Trustable Communication Between Mathematics Systems[*]

Jacques Carette, William M. Farmer, and Jérémie Wajs[†]

January 21, 2011

Abstract

This paper presents a rigorous, unified framework for facilitating communication between mathematics systems. A mathematics system is given one or more interfaces which offer deductive and computational services to other mathematics systems. To achieve communication between systems, a client interface is linked to a server interface by an asymmetric connection consisting of a pair of translations. Answers to requests are trustable in the sense that they are correct provided a small set of prescribed conditions are satisfied. The framework is robust with respect to interface extension and can process requests for abstract services, where the server interface is not fully specified.

Keywords: Mechanized mathematics, computer theorem proving, computer algebra, intersystem communication, knowledge representation, mathematical knowledge management.

[*]This research was supported by Bell Canada and MITACS. A shorter, preliminary version of this paper with the same name appeared in: T. Hardin and R. Rioboo, eds., *CALCULEMUS 2003 (11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Rome, Italy, September 2003)*, pp. 58–68, Aracne, Rome, Italy, 2003.

[†]Address: Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada. E-mail: `carette@mcmaster.ca`, `wmfarmer@mcmaster.ca`, `wajs@cas.mcmaster.ca`.

$$\begin{tikzcd}
A\text{-problem} \arrow[r, "\text{translation}"] \arrow[d, "f"'] & B\text{-problem} \arrow[d, "B\text{-service}"] \\
A\text{-answer} & B\text{-answer} \arrow[l, "\text{translation}"']
\end{tikzcd}$$

Figure 1: The basic communication problem

1 Introduction

Current mechanized mathematics systems (MMSs), by and large, fall into one of three camps: numerics-based (like Matlab, Octave, Scilab, etc), symbolic (Maple, Mathematica, MuPAD, etc), and theorem provers (Coq, HOL, IMPS, Isabelle, Nqthm, Nuprl, Otter, PVS, etc). Each has its strong points, although many are more often bemoaned for their weaknesses. These weaknesses are all the more frustrating for users as one system's weakness is frequently another's strength. An increasing majority of users are becoming agnostic in their choice of MMSs, worrying more about getting a particular task done than whether one übersystem can do it all. Furthermore, it is important to remark that the expertise needed to build each kind of system is markedly different for all three flavors. Although there have been some efforts at making some of these MMSs broader, familiarity with them quickly dispels any notion that this dabbling is particularly successful. A wiser approach, at least in the medium term, is to construct a larger system out of trusted specialized pieces.

In simple terms, the problem we wish to address, illustrated in Figure 1, is the following: if system A needs access to a certain functionality f which it does not currently implement, but a service providing this functionality is offered by system B, then A should be able to send a request to B containing a translation of its exact problem into the language of B, wait for B to perform the service, and then finally receive an answer in its own language.

Informally, we wish to think of "perform f" as a request, the pair of translations above as a connection, and the set of available functions from B-problems to B-answers as B's services. We then want to assert that *meaningful communication* happens when the diagram above commutes.

In this paper we present a unified framework which clearly defines these various concepts (*interfaces*, *services*, *connections*, *requests*, and *answers*) in precise mathematical terms. The overarching concern is that of *trust*: when one system requests a service from another, can it trust the result it gets back? Certainly any system which purports to be trustable must also insist that any communication it makes to another system satisfies the same requirements. We have not generally addressed the concept of *usefulness* of the resulting communication, as we are not aware of any generally accepted mathematical definition of that concept.

1.1 Useful Communication

Certainly examples of *useful* communication between systems abound! Commercial system builders are definitely convinced of this fact, as evidenced by Mathematica's J/Link, Maple's Matlab package, Matlab's Symbolic Toolbox, and so on.

For example, polynomial arithmetic is frequently a necessary step in a proof; typical theorem provers will, at best, implement this using rewrite rules, which are at least an order of magnitude slower than implementations by Computer Algebra Systems (CASs) [8]. In the opposite direction, closed-form integration of even simple expressions containing parameters involves complex algorithms but also complex side conditions which must be verified, forcing a CAS to call a theorem prover (see [1] and the references therein).

1.2 Obstacles

We consider old obstacles (issues of transport and syntax) to be essentially solved by common technologies (TCP/IP, sockets, XML, etc). What remains to be solved adequately is the problem of semantics. Referring back to Figure 1, it should be clear that describing each arrow, in all cases and for all possible services, is nontrivial. To achieve our aim of *trustability*, this issue is inescapable. To a lesser extent, there is also a problem of interpretability: even if the answer makes sense in system A, is it "the" answer? The notion of "the" answer in a theorem proving system is qualitatively different than in a system centered on numerical analysis, even though both are rigorously and uniquely defined.

1.3 Organization of the Paper

The rest of the paper is organized as follows: We look at previous related proposals in section 2. We give definitions for the underlying theory necessary to the presentation of our framework in section 3. In section 4, we give a simple framework for communication between MMSs. Section 5 illustrates the simple framework with an example involving decision procedures.

In section 6, we discuss additional obstacles to achieving communication in real cases, and show how to refine the framework presented in section 4 to address some of these obstacles. Section 7 illustrates the refined framework with a substantial example of communication between a theorem prover and a computer algebra system. Section 8 discusses the specification of requests and services. Finally, we conclude in section 9.

2 Previous Proposals

Several attempts at addressing the problem of communication between MMSs have been made. We can classify them into two categories: the first category consists of work that attempts to deal with the problem in general. The second category consists of *ad hoc* solutions. We review important members of each category below.

2.1 General Solutions

The OpenMath project [19] claims to provide a common platform for communication between various mathematics systems. However, while it provides a common syntax, as well as a set of common reference semantics in the form of "content dictionaries", it fails in our view to specify a machine readable semantics for that syntax, which is a major drawback when trying to make mathematics systems based on different logics communicate. In other words, there are too many implicit assumptions behind OpenMath's version of semantics (as embodied in its content dictionaries) for it to apply outside the narrow (but useful) realm of standard operations between the standard CASs.

OMDoc [31] constitutes a refinement to the OpenMath approach: it recognizes the need for deeper semantics, and introduces them through a notion of *theories*. However, OMDoc does not seem to address the actual mechanics of getting different systems to communicate as much as it provides a common language (syntax + semantics) for them to do so. Nevertheless, OMDoc could be extended to handle the concepts of our framework: interfaces, services, connections, requests, and answers.

The Ω-MKRP [30] approach argues that explicit proofs are needed and that "external" systems cannot be trusted. This seems very impractical.

The OMSCS (Open Mechanized Symbolic Computation Systems) [9] work provides an architecture used to formally specify automated theorem provers and CASs and to formally integrate them. However, it does not seem to address the issues of trust or extending theories.

Armando and Zini's Logic Broker Architecture [3], defines a general framework for communication between MMSs. This approach is conceptually very similar to ours. It defines interfaces for MMSs and uses a *Logic Broker (LB)* to achieve communication between systems. The LB includes facilities for translation of requests and meaning-preserving translation of answers (thus addressing the question of trust), as well as (in theory) a logical specification matcher to match requests to services offered. However, we believe that this architecture does not support extending theories well, which we will show can be achieved effectively by our approach.

The new European MOWGLI project [4], which aims at providing a common machine-understandable (semantics-based) representation of mathematical knowledge and a platform to exploit it, likely fits here too.

2.2 Ad-hoc Approaches

In many such cases in the literature, only unidirectional cooperation exists: one system acts as a master, generating requests, while the other one serves as a slave, fulfilling those requests. This includes Howe's work on embedding an HOL theory into Nuprl [25], Ballarin and Paulson's work on using the Σ^{it} library for proofs in Isabelle [6, 8], and Ballarin, Homann, and Calmet's work on an interface between Isabelle and Maple [7]. Ballarin and Paulson's work clearly identifies the issue of trust, and distinguishes between trustable results, for which a formal proof exists, and ad hoc results, based on approximations.

Another more complex *ad hoc* case, intended for bidirectional cooperation, is Harrison and Théry's work on combining HOL and Maple [24]. Similarly to Ballarin and Paulson, they classify the systems by *degree of trust*, for example trusting results proved by HOL while checking results given by Maple.

All these *ad hoc* solutions have the major drawback of not seeking generality. Howe, for instance, does not attempt to make HOL and Nuprl *communicate* as much as he attempts to *embed* an HOL theory into Nuprl. Why should the machinery for HOL be duplicated in Nuprl when it already exists in HOL itself? In addition, this approach is not valid when the system to be integrated is a black box. Our approach enables one MMS to use another MMS's services without, first, having to reproduce them, and second, having to know in detail how they work. We will show how it addresses the issue of trust, and eliminates the need to verify every single result (which can be painfully burdensome).

3 Biform Theories and Translations between Them

At the heart of this work lies the notion of a "biform theory", which is the basis for FFMM, a Formal Framework for Managing Mathematics [22]. Informally, a biform theory is simultaneously an axiomatic and an algorithmic theory. Most of the definitions given here are simplified versions of definitions given in [22].

3.1 Logics

A *language* is a set of typed expressions. The types include $*$, which denotes the type of truth values. A *formula* is an expression of type $*$. For a formula A of a language L, $\neg A$, the negation of A, is also a formula of L. A *logic* is a set of languages with a notion of logical consequence. If $\mathbf{K}$ is a logic, L is a language of $\mathbf{K}$, and $\Sigma \cup \{A\}$ is a set of formulas of L, then $\Sigma \models_{\mathbf{K}} A$ means that A is a logical consequence of Σ in $\mathbf{K}$.

3.2 Transformers and Formuloids

Let L_i be a language for $i = 1, 2$. A *transformer* Π from L_1 to L_2 is an algorithm that implements a partial function $\pi : L_1 \rightharpoonup L_2$. For $E \in L_1$, let $\Pi(E)$ mean $\pi(E)$, and let $\mathsf{dom}(\Pi)$ denote the domain of π, i.e., the subset of L_1 on which π is defined. For more on transformers, see [21, 22].

A *formuloid* of a language L is a pair $\theta = (\Pi, M)$ where:

(1) Π is a transformer from L to L.

(2) M is a function that maps each $E \in \mathsf{dom}(\Pi)$ to a formula of L.

M is intended to give the *meaning* of applying Π to an expression E. $M(E)$ usually relates the input E to the output $\Pi(E)$ in some way; for many transformers, $M(E)$ is the equation $E = \Pi(E)$, which says that Π transforms E into an expression with the same value as E itself.

The *span* of θ, written $\mathsf{span}(\theta)$, is the set

$$\{M(E) \mid E \in \mathsf{dom}(\Pi)\}$$

of formulas of L. Thus a formuloid has both an *axiomatic meaning*—its span—and an *algorithmic meaning*—its transformer. The purpose of its span is to assert the truth of a set of formulas, while its transformer is meant to be a deduction or computation rule.

3.3 Biform Theories

A *biform theory* is a triple $T = (\mathbf{K}, L, \Gamma)$ where:

(1) $\mathbf{K}$ is a logic called the *logic* of T.

(2) L is a language of $\mathbf{K}$ called the *language* of T.

(3) Γ is a set of formuloids of L called the *axiomoids* of T.

The *span* of T, written $\mathsf{span}(T)$, is the union of the spans of the axiomoids of T, i.e.,

$$\bigcup_{\theta \in \Gamma} \mathsf{span}(\theta).$$

A is an *axiom* of T if $A \in \mathsf{span}(T)$. A is a *(semantic) theorem* of T, written $T \models A$, if

$$\mathsf{span}(T) \models_{\mathbf{K}} A.$$

A *theoremoid* of T is a formuloid θ of L such that, for each $A \in \mathsf{span}(\theta)$, $T \models A$. Obviously, each axiomoid of T is also a theoremoid of T. An axiomoid is a generalization of an axiom; an individual axiom A (in the usual sense) can be represented by an axiomoid (Π, M) such that $\mathsf{dom}(\Pi) = \{A\}$ and $M(A) = A$.

T can be viewed as simultaneously both an *axiomatic theory* and an *algorithmic theory*. The axiomatic theory is represented by

$$T_{\rm axm} = (\mathbf{K}, L, \{M(E) \mid (\Pi, M) \in \Gamma \text{ for some } \Pi \text{ and } E \in \mathsf{dom}(\Pi)\}),$$

and the algorithmic theory is represented by

$$T_{\rm alg} = (\mathbf{K}, L, \{\Pi \mid (\Pi, M) \in \Gamma \text{ for some } M\}).$$

Let $T_i = (\mathbf{K}, L_i, \Gamma_i)$ be a biform theory for $i = 1, 2$. T_2 is an *extension* of T_1, written $T_1 \le T_2$, if $L_1 \subseteq L_2$ and $\Gamma_1 \subseteq \Gamma_2$. T_2 is a *conservative extension* of T_1, written $T_1 \trianglelefteq T_2$, if $T_1 \le T_2$ and, for all formulas A of L_1, if $T_2 \models A$, then $T_1 \models A$. Note that $\le$ and $\trianglelefteq$ are partial orders.

3.4 Translations and Interpretations

Let $\mathbf{K}_i$ be a logic and $T_i = (\mathbf{K}_i, L_i, \Gamma_i)$ be a biform theory for $i = 1, 2$. A *translation* from T_1 to T_2 is a transformer Φ from L_1 to L_2 that:

(1) Respects types, i.e., if E_1 and E_2 are expressions in L_1 of the same type and $\Phi(E_1)$ and $\Phi(E_2)$ are defined, then $\Phi(E_1)$ and $\Phi(E_2)$ are also of the same type.

(2) Respects negation, i.e., if A is a formula in L_1 and $\Phi(A)$ is defined, then $\Phi(\neg A) = \neg\Phi(A)$.

T_1 and T_2 are called the *source theory* and the *target theory* of Φ, respectively. Φ is *total* if $\Phi(E)$ is defined for each $E \in L_1$. Φ *fixes* a language L if $\Phi(E) = E$ for each $E \in L$.

An *interpretation* of T_1 in T_2 is a total translation Φ from T_1 to T_2 such that, for all formulas $A \in L_1$, if $T_1 \models A$, then $T_2 \models \Phi(A)$. An interpretation thus maps theorems to theorems. (Since any translation respects negation, an interpretation also maps negated theorems to negated theorems.) A *retraction* from T_2 to T_1 is an interpretation Φ of T_2 in T_1 such that $T_1 \leq T_2$ and Φ fixes L_1.

Lemma 3.1 *Let Φ_1 be a retraction from T_2 to T_1 and Φ_2 be a retraction from T_3 to T_2. Then $\Phi_1 \circ \Phi_2$ is a retraction from T_3 to T_1.*

Proof Let $\Phi = \Phi_1 \circ \Phi_2$. We first need to prove that Φ is an interpretation. Φ is clearly total. Assume $T_3 \models A$. Then $T_2 \models \Phi_2(A)$ since Φ_2 is an interpretation of T_3 in T_2. In turn, $T_1 \models \Phi_1(\Phi_2(A))$, i.e., $T_1 \models \Phi(A)$ since Φ_1 is an interpretation of T_2 in T_1. Hence, Φ is an interpretation of T_3 in T_1.

By transitivity of $\leq$, since $T_1 \leq T_2$ and $T_2 \leq T_3$, $T_1 \leq T_3$.

Finally, we need to prove that Φ fixes L_1. Let $E \in L_1 \subseteq L_2 \subseteq L_3$. $\Phi_2(E) = E$ since Φ_2 is a retraction from T_3 to T_2 and $E \in L_2$. Similarly, $\Phi_1(\Phi_2(E)) = \Phi_1(E) = E$ since Φ_1 is a retraction from T_2 to T_1 and $E \in L_1$. Hence $\Phi(E) = E$ and Φ fixes L_1. $\Box$

Proposition 3.2 *If Φ is a retraction from T_2 to T_1, then $T_1 \trianglelefteq T_2$.*

Proof Let A be a formula of the language of T_1 such that $T_2 \models A$. We must show that $T_1 \models A$. By definition, (1) Φ is an interpretation of T_2 in T_1 and (2) Φ fixes the language of T_1. (1) implies that $T_1 \models \Phi(A)$, and (2) implies $\Phi(A) = A$. Therefore, $T_1 \models A$. $\Box$

4 A Simple Communication Framework

We now present a simple communication framework, based on the theoretical notions presented in the previous section, that addresses the problem presented in Figure 1. The framework formalizes the notions we mentioned in the introduction: interface, service, connection, request, and answer. As we will show after this section, the framework does not address some important obstacles to effective communication between MMSs. A refined framework, which is more practical and which generalizes this simple framework, is presented in section 6.

An *interface* is a pair $I = (T, \mathcal{S})$ where:

(1) T is a biform theory called the *theory* of I.

(2) $\mathcal{S}$ is a set of theoremoids of T called the *services* of I.

As a theoremoid of T, a service of I is a formuloid whose span is a set of theorems of T and whose transformer is a sound deduction or computation rule for T.

Let $I_i = (T_i, \mathcal{S}_i)$ be an interface for $i = 1, 2$. A *connection* from I_1 to I_2 is a pair $C = (\mathsf{export}, \mathsf{import})$ where export is a translation from T_1 to T_2, and import is an interpretation of T_2 in T_1. I_1 and I_2 are respectively called the *client interface* and the *server interface* of C. export is for transporting problems from T_1 to T_2; it need not be meaning preserving. import transports solutions from T_2 to T_1; it must be meaning preserving.

An *informed request* is a triple $R = (C, E, \theta)$ where:

(1) $C = (\mathsf{export}, \mathsf{import})$ is a connection from $I_1 = (T_1, \mathcal{S}_1)$ to $I_2 = (T_2, \mathcal{S}_2)$.

(2) E is an expression of the language of T_1.

(3) $\theta = (\Pi, M) \in \mathcal{S}_2$.

The reason to call such a request *informed* is that it depends not only on the server interface I_2 but on the service θ as well: we assume that the client interface I_1 "knows" about θ. We will come back to this point in section 8.

If

$$A = (\mathsf{import} \circ M \circ \mathsf{export})(E)$$

is defined, it is the *answer* to R; otherwise the answer to R is undefined. When A is defined, it is a theorem:

Proposition 4.1 *Let R and A be as above. If A is defined, then $T_1 \models A$.*

$$\begin{tikzcd} E \arrow[r, "\mathsf{export}"] \arrow[d, "?"'] & E' \arrow[d, "\theta"] \\ A & M(E') \arrow[l, "\mathsf{import}"] \end{tikzcd}$$

Figure 2: Communication between two MMSs

Proof Assume A is defined. Since θ is a theoremoid of T_2, $T_2 \models (M \circ \mathsf{export})(E)$, and then since import is an interpretation of T_2 in T_1, $T_1 \models A$. $\Box$

Note that, if C and θ are not chosen well, A may be a useless theorem such as true or $E = E$.

The basic problem (Figure 1) is now addressed as shown in Figure 2. All that is necessary to perform this type of communication are interfaces for both systems and a connection between the two interfaces.

This takes care of the question of *trust* (should A believe the answer it receives from B?), so crucial to the general problem at hand. Whether an answer is correct depends on whether a translation is an interpretation and a service is a theoremoid. Thus an answer is trustworthy if the mechanisms for verifying interpretations and theoremoids are trustworthy.

Note also at this point that a given system may have many interfaces, each containing only one or a few services of that system. This approach allows us to consider trustable subsystems within a system and to use those subsystems in trustable communication. For example, while a result given by Maple cannot be fully trusted in general, many subparts of Maple are well encapsulated and could be proved correct.

5 An Example using Decision Procedures

Suppose S_{hol} is a higher-order interactive theorem proving system with several implemented theories including COF, a theory of a complete ordered field. COF has one model up to isomorphism, namely, the real numbers with the usual operations such as $+$, $*$, and $<$. An exceedingly rich theory, COF is adequate for developing real analysis. Suppose also that S_{fol} is a first-order automated theorem proving system with several implemented theories equipped with decision procedures including PA, a theory of first-order Peano arithmetic. The theoremoids of PA include θ_+, a decision procedure for additive number theory (Presburger arithmetic), and θ_*, a decision procedure for multiplicative number theory (sometimes called Skolem arithmetic). The framework outlined above can be used to give $S_{\rm hol}$ access to the decision procedures in $S_{\rm fol}$.

Let $I_1 = (\mathsf{COF}, \mathcal{S}_1)$ be an interface of $S_{\rm hol}$ and $I_2 = (\mathsf{PA}, \mathcal{S}_2)$ with $\{\theta_+, \theta_*\} \subseteq \mathcal{S}_2$ be an interface of $S_{\rm fol}$. Also let $C = (\mathsf{export}, \mathsf{import})$ be the connection from I_1 to I_2 where export translates "first-order natural number formulas" of COF to formulas of PA and import is a standard interpretation of PA in COF. export is not an interpretation because, unlike PA, COF satisfies the axioms of second-order Peano arithmetic, and thus there are theorems of COF that export maps to nontheorems of PA. (export is also not an interpretation because it is not a total translation.) The interpretation import exists because COF satisfies the axioms of second-order Peano arithmetic and hence also satisfies the axioms of first-order Peano arithmetic.

C offers a way of deciding in COF many statements about the natural numbers using the two decision procedures θ_+ and θ_*, both of which are nontrivial to implement. As an illustration, if the request $R = (C, E, \theta_+)$ is made where E is a formula in COF that expresses the Presburger theorem

$$\forall\, a, b, c : \mathbf{N}\ .\ a \equiv b \ \mathsf{mod}\ n \Leftrightarrow a + c \equiv b + c \ \mathsf{mod}\ n,$$

then the answer might be something like $E \Leftrightarrow \mathsf{true}$.

We can continue the example by supposing that the implemented theories of $S_{\rm fol}$ also includes RCF, a formalization of the first-order theory of real closed fields (see [13] for a precise description of this theory). The theoremoids of RCF include $\theta_{\rm tarski}$, a decision procedure for RCF.[1] Let $I_3 = (\mathsf{RCF}, \mathcal{S}_3)$ with $\{\theta_{\rm tarski}\} \subseteq \mathcal{S}_3$ be an interface of $S_{\rm fol}$. Since a complete ordered field is also a real closed field, it is possible to define a connection from I_1 to I_3 which will enable a wide range of algebraic statements about the real numbers expressed in COF to be decided using the decision procedure $\theta_{\rm tarski}$.

6 A Refined Communication Framework

There are several obstacles to effectively employing the simple framework presented in section 4. In this section, three obstacles involving connections are addressed.

[1]A. Tarski proved using quantifier elimination that RCF is a decidable theory (see [36]). Tarski's decision procedure and other more efficient decision procedures for RCF are computationally infeasible for large inputs.

6.1 Obstacles involving Connections

The first obstacle is that constructing connections between interfaces is a challenging task, especially when the biform theories of the interfaces are based on different logics. The export translation of a connection must satisfy a syntactic condition, but the import interpretation must satisfy both a syntactic and semantic condition. As a general principle, it is easier to construct a translation or interpretation Φ if the "primitive basis" of its source theory T_1 (the primitive symbols and axiomoids of T_1) is small. In this case, the translation of a complex expression of T_1 may simply be a composition of the translations of its primitive parts.

The second obstacle is that translating an expression E using the export translation or the import interpretation of a connection may result in an expression much larger than E. As a general principle, it is easier to construct a translation or interpretation Φ without this kind of size explosion if its target theory T_2 contains a rich set of defined symbols. In this case, the translation of an expression of T_1 could be made concise using the defined symbols of T_2.

The third obstacle is that the theory S of an MMS behind the biform theory T of an interface is likely to be enriched with defined symbols over time. Defining a symbol in S will have the effect of extending T to a new theory T'. However, an interpretation Φ of T will not be an interpretation of T' because Φ will not be defined on expressions of T' containing the new defined symbol. As a result, any connection to an interface of the form $(T, \mathcal{S})$ will be broken by the definition of the new symbol.

6.2 Conservative Stacks

These three obstacles can be addressed by using a "conservative stack" in place of a biform theory in the definition of an interface. Interface, connection, informed request, and answer are redefined. The resulting refined framework is a generalized version of the simple framework.

A *conservative stack* is a pair $\Sigma = (\tau, \rho)$ of sequences where:

(1) $\tau = \langle T_0, \ldots, T_n \rangle$ is a finite sequence of biform theories such that, for all i with $0 \leq i < n$, $T_i \leq T_{i+1}$. T_n is called the *theory* of Σ.

(2) $\rho = \langle \Phi_1, \ldots, \Phi_n \rangle$ is a finite sequence of translations such that, for all i with $0 < i \leq n$, Φ_i is a retraction from T_i to T_{i-1}.

Notice that, by Proposition 3.2, the sequence ρ of retractions implies that τ is a "stack" of conservative extensions, i.e., $T_0 \trianglelefteq \cdots \trianglelefteq T_n$. The conservative stack Σ represents a conservative "development" from the biform theory T_0 to its extension T_n.

An *interface* is a pair $I = (\Sigma, \mathcal{S})$ where Σ is a conservative stack and $\mathcal{S}$ is a set of theoremoids of the theory of Σ called the *services* of I.

Let $I_i = ((\tau_i, \rho_i), \mathcal{S}_i)$ be an interface with $\tau_i = \langle T^i_0, \ldots, T^i_{n_i} \rangle$ for $i = 1, 2$. A *connection* C from I_1 to I_2 is a pair $(\mathsf{export}, \mathsf{import})$ where:

(1) export is a translation from U^1 to V^2.

(2) import is an interpretation of U^2 in V^1.

(3) U^1 and V^1 are members of τ_1.

(4) U^2 and V^2 are members of τ_2

Let Φ_i be the composition of elements of ρ_i from $T^i_{n_i}$ to U^i for $i = 1, 2$. By Lemma 3.1, Φ_i is a retraction from $T^i_{n_i}$ to U^i for $i = 1, 2$. Then

$$(\mathsf{export} \circ \Phi_1, \mathsf{import} \circ \Phi_2)$$

is a connection from $(T^1_{n_1}, \mathcal{S}_1)$ to $(T^2_{n_2}, \mathcal{S}_2)$ in the simple framework even if $U^1 \neq V^1$ or $U^2 \neq V^2$.

An *informed request* is a triple $R = (C, E, \theta)$ where:

(1) C is a connection from I_1 to I_2 as defined above.

(2) E is an expression of the language of $T^1_{n_1}$, the theory of I_1.

(3) $\theta = (\Pi, M) \in \mathcal{S}_2$.

If

$$A = (\mathsf{import} \circ \Phi_2 \circ M \circ \mathsf{export} \circ \Phi_1)(E)$$

is defined (where Φ_1 and Φ_2 are defined as above), it is the *answer* to R; otherwise the answer to R is undefined. When A is defined, it is a theorem:

Proposition 6.1 *Let R and A be as above. If A is defined, then $V^1 \models A$.*

Proof Assume that A is defined. Since θ is a theoremoid of $T^2_{n_2}$, the theory of I_2, $T^2_{n_2} \models (M \circ \mathsf{export} \circ \Phi_1)(E)$, and since Φ_2 is a retraction from $T^2_{n_2}$ to U^2, $U^2 \models (\Phi_2 \circ M \circ \mathsf{export} \circ \Phi_1)(E)$. Since import is an interpretation of U^2 in V^1, we conclude that $V^1 \models A$. $\Box$

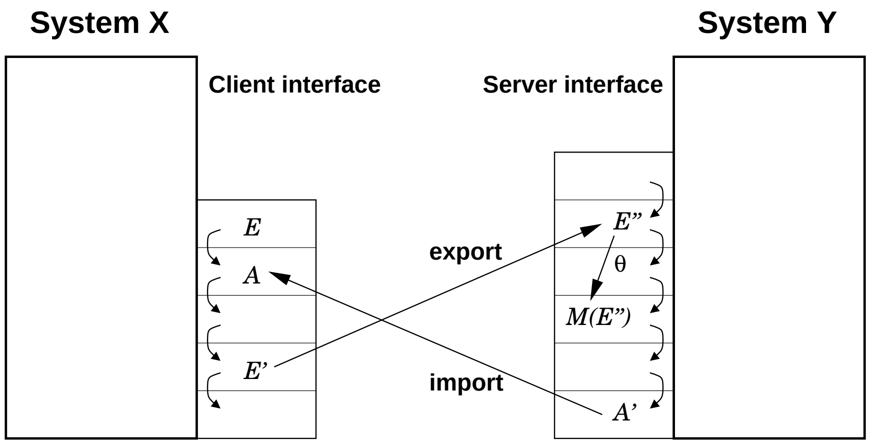

Figure 3: The refined communication framework

The refined communication framework is illustrated in Figure 3. It shows two interfaces that are connected. The client interface has a conservative stack

$$(\langle T_0^1, \ldots, T_4^1 \rangle, \langle \Phi_1^1, \ldots, \Phi_4^1 \rangle),$$

and the server interface has a conservative stack

$$(\langle T_0^2, \ldots, T_5^2 \rangle, \langle \Phi_1^2, \ldots, \Phi_5^2 \rangle).$$

The connection is $C = (\mathsf{export}, \mathsf{import})$ where export is a translation from T_1^1 to T_4^2 and import is an interpretation of T_0^2 in T_3^1. The small arrows depict the retractions $\Phi_1^1, \ldots, \Phi_4^1, \Phi_1^2, \ldots, \Phi_5^2$. Let $R = (C, E, \theta)$ be the informed request where E is an expression of T_4^1 and $\theta = (\Pi, M)$ is a service of the server interface. Then the answer to R is

$$A = (\mathsf{import} \circ \Phi_1^2 \circ \Phi_2^2 \circ \Phi_3^2 \circ \Phi_4^2 \circ \Phi_5^2 \circ M \circ \mathsf{export} \circ \Phi_2^1 \circ \Phi_3^1 \circ \Phi_4^1)(E),$$

provided A is defined. (Since $M(E'')$ resides in the language of T_2^2, $\Phi_1^2 \circ \cdots \circ \Phi_5^2$ could be replace with $\Phi_1^2 \circ \Phi_2^2$.)

The refined framework facilitates the construction of a translation or interpretation Φ between two interfaces I_1 and I_2 by allowing the source theory of Φ to be chosen from the lower part of the conservative stack of I_1 and the target theory of Φ to be chosen from the upper part of the conservative stack of I_2 (addressing the first and second obstacles discussed above). If a conservative stack Σ is extended to a larger conservative stack Σ', then Σ can be freely replaced with Σ' without compromising any existing interfaces or connections (addressing the third obstacle).

7 Examples of Interaction with Computer Algebra Systems

Current documented uses of a CAS by a theorem prover are almost all of the get-and-check variety: call a CAS to get an answer, and then prove that the answer is correct. For many problems, like factoring of large integers or polynomials, this is efficient and effective. The exceptions are the systems *Analytica* [16] and *Theorema* [12] which are theorem provers built on top of Mathematica; this makes them extremely powerful and singularly untrustworthy, an intriguing combination. We seek a much wider applicability, and thus we want to be able to call a computer algebra system to perform correct computations where verification can be just as expensive as the computation itself. With conservative stacks of biform theories, especially when the theories are aimed at a particular subsystem of a CAS, this is possible.

The case where a computer algebra system can profitably call a theorem proving system has been documented in the literature already [1, 2]. Currently the main uses of a theorem prover by a CAS is in checking side-conditions for the applicability of certain theorems and algorithms, side-conditions which are frequently ignored in current CASs. One could cynically observe that this is a futile exercise as computer algebra systems are widely documented as being inconsistent [11, 20, 27, 28, 35], and that creating interpretations with inconsistent target theories is trivial. On the other hand, it is also the case that many of these inconsistencies stem from the lack of verification of simple side conditions necessary for the applicability of a particular method. There is thus a reasonable hope that this kind of communication could become the crucial step in making some of the functionality present in computer algebra systems trustworthy.

There exists a very good rule-of-thumb to understand where current computer algebra systems are or are not trustworthy: algebra versus analysis. Arithmetic computations for integers, rationals, polynomials, algebraic numbers, finite fields, formal series, and matrices over any of these domains are in modularized sub-systems with well-understood interfaces and completely trustable theories. This extends to computations of such things as normal forms for presentations of left R-modules over differential rings (via Gröbner basis computations) [14, 15] or generating systems of holonomic equations satisfied by multivariate generating functions associated to attribute grammars for combinatorial structures [23, 32]. On the other hand computations like differentiation, integration (quadrature), summation, and solving differential equations, not only have fuzzy interfaces, but also have theoretical underpinnings that are still in a heavy state of flux. To be precise, let us consider one example: Risch integration [33]. This celebrated algorithm is completely and firmly grounded in the algebraic theory of differential fields, which we believe could be fully formalized. However, it is also well-known that this theory does not respect basic features of analysis (see [26] and the appendix of [29]) which implies that it is an inaccurate model for the task at hand, as integration is a firmly analytic concept!

7.1 Computations during Proofs

We will consider two examples, one used by Armando and Zini in [3], as well as one used by Ballarin [6, 8], but in the context of our framework. For brevity we will assume that the reader is familiar with these papers and will use their notation without further introduction.

The problem used by Armando and Zini, originally from [10], is to show the unsatisfiability of

$$\mathsf{ms}(c) + \mathsf{ms}(a)^2 + \mathsf{ms}(b)^2 \geq \mathsf{ms}(c) + \mathsf{ms}(b)^2 + 2 * \mathsf{ms}(a)^2 * \mathsf{ms}(b) + \mathsf{ms}(a)^4$$

using the lemma

$$\forall X.(\mathsf{ms}(X) > 0),$$

where ms is some unknown but fixed function. The proof, as presented, uses many standard techniques in theorem proving, but also relies on the fact that

$$\mathsf{ms}(a)^4 + 2 * \mathsf{ms}(a)^2 * \mathsf{ms}(b) - \mathsf{ms}(a)^2 = \mathsf{ms}(a)^2 * (\mathsf{ms}(a)^2 + 2 * \mathsf{ms}(b) - 1),$$

which is readily computed by any computer algebra system. It is important to note that this only relies on polynomial arithmetic and not on a factorizer that provably decomposes a polynomial into irreducible factors.

In our framework, a straightforward translation of Armando and Zini's example would have system A be RDL and system B be CoCoA (although it could just as easily have been Maple or Mathematica). We can then set up a biform theory T_1 for RDL's notion of (factored) polynomials, and T_2 for the same theory in CoCoA. Here T_1 would be a mostly axiomatic theory, with axiomoids that state the properties of factoring but do not include a factoring transformer. T_2 would be the a similar axiomatic theory, but with an axiomoid θ that contains a transformer for *factoring*. See Chapters 4 and 5 of Ballarin's PhD thesis [6] for details on these axiomatizations. It is important to note that T_2 is part of the *interface* to the CAS, and does not necessarily have a computerized representation; in other words one has to trust that the axiomatization of T_2 and its effective implementation are consistent. The interface $I_1 = (T_1, \emptyset)$ does not need any services, but the interface $I_2 = (T_2, \{\theta\})$ needs to contain at least θ. Since we are assuming that the theories T_1 and T_2 are semantically very similar, the connection components export and import are quite easy to formulate as they can be essentially syntactic in nature.[2]

The informed request associated to this query would include an expression like

$$\mathsf{ms}(a)^4 + 2 * \mathsf{ms}(a)^2 * \mathsf{ms}(b) - \mathsf{ms}(a)^2$$

in the language of T_1, and export would translate it to an expression of the language of T_2. In particular, θ could be $(\mathsf{factor}, \lambda x.\mathsf{Factor}(x) = \mathsf{factor}(x))$. Thus upon evaluation and interpretation, we would get

$$\mathsf{Factor}(\mathsf{ms}(a)^4 + 2 * \mathsf{ms}(a)^2 * \mathsf{ms}(b) - \mathsf{ms}(a)^2) = \mathsf{ms}(a)^2 * (\mathsf{ms}(a)^2 + 2 * \mathsf{ms}(b) - 1)$$

as an answer. Above, we are using the convention (from Maple) that Factor describes the problem of factorization where factor describes the algorithm that implements a solution to this problem; this is frequently described as saying that Factor is a noun and factor is a verb, but both represent the same semantic concept. It is rather unfortunate that this convention has yet to be described in the literature, and that this noun/verb relationship between mathematical concepts and their implementations is only very sparsely implemented (in any CAS).

The second problem is one involving BCH codes, where one needs to do factorizations into irreducibles (over $\mathbf{F}_2[X]$) as well as performing Gaussian elimination (over $M_n(\mathbf{F}_2)$). The complete problem is laid out in Chapter 5 of [6]. In this case, system A is Isabelle and B is Σ^{it}, a library built on top of the CAS *Axiom*. Although the theories, services, and interfaces are more complex, they are essentially built exactly as outlined in the previous paragraph. The biggest difference here is that in order to prove that a particular polynomial is a unique and minimal solution to the problem at hand, one would need to essentially redo all the steps of a Gaussian elimination, but in prover A's calculus; this is considerably more difficult that the usual get-and-check approach. More details can be found in section 5.3.3 of [6]. It is worthwhile to observe that conservative stacks are effectively used in the theories T_1 and T_2 associated to these problems, and that these levels

[2]We note that this is a rare occurrence, and it only seems to happen for old and well-understood mathematics which has been axiomatized and implemented in a "classical" manner.

$$\begin{tikzcd} E \arrow[r, "\mathsf{export}"] \arrow[d, "\theta_1"'] & E' \arrow[d, "\theta_2"] \\ A & M(E') \arrow[l, "\mathsf{import}"] \end{tikzcd}$$

Figure 4: The specification matching problem

manifest themselves in the communication (see section 5.3.5 for details). No proof that import is actually an interpretation is provided, but so much care was taken in setting up T_1 and T_2 that we would be surprised if it were not the case.

8 Specifying Requests and Services

Until now, we assumed that system A "magically" knows that it wants to use service θ of system B. However, in a more general setting, one would want to specify a request (like *evaluate this computation*), and pass that specification on to some entity able to match it to an available service.

Thus, instead of dealing with services of I_2, we need to deal with some specification S corresponding to some function (transformer) $f : L_1 \rightarrow L_1$ associated with a "virtual service" θ_1. Given S, the task then becomes one of *finding* an informed request such that our communication diagram commutes. In theory, this is what we understand Armando and Zini's LS Matcher [3] is intended to do, although its task is never defined precisely.

Let us define a *reachable service* as a theoremoid θ_2 of T_2 that can be given a complete specification in some meta-language Spec. We could, for example, use CASL [5], Z [37], or Specware [34] for this task. In other words, we wish to define services (and requests) implicitly, allowing nonconstructive definitions as well. Note that we specifically exclude theoremoids that cannot be finitely axiomatized in Spec. Symmetrically to the notion of a reachable service, we define a *(brokered) request* as a pair (E, θ_1) where E is an expression of L_1 and θ_1 is a virtual service of I_1 which can be specified completely in Spec. (Notice that the request does not contain a connection.)

We then need to solve the specification matching problem: given a pair (S_1, S_2) of specifications for θ_1 and θ_2, does there exist a connection C such that our communication diagram commutes?

Even in the simplest possible case where both systems are the same, this problem can still be quite difficult unless great pains are taken to specify each system's services in a very uniform manner. However the situation is far from hopeless: even though there are many different ways to specify that, for example, a particular function is a primality verification function (or an implementation thereof), the task of deciding that two such specifications are equivalent is considerably simpler than actually providing a provably correct implementation!

One should also note the clear applicability of this framework to the issue of mathematically-oriented *Web services* [17, 18], where one can use SOAP and other standard XML encodings for the syntax while retaining the semantics of the specifications.

9 Conclusion

In this paper we have presented a mathematically rigorous framework for communicating mathematics between MMSs. This framework gives precise meanings to notions such as *(biform) theories*, *interfaces*, *services*, *connections*, *requests*, and *answers*. It addresses the issue of trust, which has been identified as a central issue in intersystem communication in related papers, by using *interpretations* (meaning-preserving translations) to communicate answers. It also provides facilities for conservatively extending theories, allowing them to evolve as needed without needing to rebuild whole new interfaces or to drastically update connections.

We have defined precisely the problem of specification of services, and of logical specification matching. We are aware that any useful implementation of the ideas detailed in this paper would need to include such a facility, and we are working in that direction.

References

[1] A. Adams, M. Dunstan, H. Gottliebsen, T. Kelsey, U. Martin, and S. Owre. Computer algebra meets automated theorem proving: Integrating Maple and PVS. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2001)*, volume 2152 of *LNCS*, pages 27–42. Springer-Verlag, 2001.

[2] Andrew A. Adams, Hanne Gottliebsen, Steve Linton, and Ursula Martin. Automated theorem proving in support of computer algebra: Symbolic definite integration as a case study. In *Proceedings of International*

Symposium on Symbolic and Algebraic Computation (ISSAC'99), pages 253–260, Vancouver, British Columbia, Canada, 1999. ACM.

[3] A. Armando and D. Zini. Interfacing computer algebra and deduction systems via the logic broker architecture. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning (CALCULEMUS-2000)*, pages 49–64. A. K. Peters, 2001.

[4] A. Asperti and B. Wegner. MOWGLI—a new approach for the content description in digital documents. In *Ninth International Conference on Electronic Resources and the Social Role of Libraries in the Future*, Autonomous Republic of Crimea, Ukraine, 2002.

[5] E. Astesiano, M. Bidoit, H. Kirchner, B. Krieg-Brückner, P. D. Mosses, D. Sannella, and A. Tarlecki. CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, 286:153–196, 2002.

[6] C. Ballarin. *Computer Algebra and Theorem Proving.* PhD thesis, Cambridge University, 1999.

[7] C. Ballarin, K. Homann, and J. Calmet. Theorems and algorithms: An interface between Isabelle and Maple. In *International Symposium on Symbolic & Algebraic Computation (ISSAC-95)*, pages 150–157, 1995.

[8] C. Ballarin and L. C. Paulson. A pragmatic approach to extending provers by computer algebra - with applications to coding theory. *Fundamenta Informaticae*, 39:1–20, 1999.

[9] P. G. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. *Fundamenta Informaticae*, 39:39–57, 1999.

[10] R. S. Boyer and J Strother Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. *Machine Intelligence*, 11:83–124, 1988.

[11] R.J. Bradford and J.H. Davenport. Towards better simplification of elementary functions. In Bernard Mourrain, editor, *ACM International Symposium on Symbolic and Algebraic Computation*, pages 15–22, 2002.

[12] Bruno Buchberger, Tudor Jebelean, Franz Kriftner, Mircea Marin, Elena Tomuta, and Daniela Vasaru. A survey of the theorema project. In *International Symposium on Symbolic and Algebraic Computation*, pages 384–391, 1997.

[13] C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland, 1990.

[14] Frédéric Chyzak. An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, 217(1-3):115–134, 2000. Formal power series and algebraic combinatorics (Vienna, 1997).

[15] Frédéric Chyzak and Bruno Salvy. Non-commutative elimination in Ore algebras proves multivariate holonomic identities. *Journal of Symbolic Computation*, 26(2):187–227, August 1998.

[16] Edmund Clarke and Xudong Zhao. Analytica-A theorem prover in mathematica. In *Automated Deduction-CADE-II*, pages 761–763, 11th International Conference on Automated Deduction, Saratoga Springs, New York, June 15-18 1992.

[17] World Wide Web Consortium. http://www.w3c.com.

[18] World Wide Web Consortium. Mathematical markup language (MathML) version 2.0.

[19] S. Dalmas, M. Gaëtano, and S. M. Watt. An OpenMath 1.0 implementation. In *International Symposium on Symbolic & Algrebraic Computation (ISSAC-97)*, pages 241–248, 1997.

[20] J.H. Davenport. Equality in computer algebra and beyond. *Journal of Symbolic Computation*, 34:259–270, 2002.

[21] W. M. Farmer and M. v. Mohrenschildt. Transformers for symbolic computation and formal deduction. In S. Colton, U. Martin, and V. Sorge, editors, *Proceedings of the Workshop on the Role of Automated Deduction in Mathematics, CADE-17*, pages 36–45, 2000.

[22] W. M. Farmer and M. v. Mohrenschildt. An overview of a Formal Framework for Managing Mathematics. *Annals of Mathematics and Artificial Intelligence*, 38:165–191, 2003.

[23] Philippe Flajolet and Bruno Salvy. Computer algebra libraries for combinatorial structures. *Journal of Symbolic Computation*, 20(5-6):653–671, 1995.

[24] J. Harrison and L. Théry. A skeptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[25] D. J. Howe. Importing mathematics from HOL into Nuprl. In J. Von Wright et al., editors, *Theorem Proving in Higher Order Logics (TPHOLs 1996)*, volume 1125 of *LNCS*, pages 267–282. Springer-Verlag, 1996.

[26] David J. Jeffrey. Integration to obtain expressions valid on domains of maximum extent. In *International Symposium on Symbolic and Algebraic Computation*, pages 34–41, 1993.

[27] D.J. Jeffrey. The importance of being continuous. *Mathematics Magazine*, 67:294–300, 1994.

[28] D.J. Jeffrey. Rectifying transformations for the integration of rational trigonometric functions. *Journal of Symbolic Computation*, 24:563–573, 1997.

[29] Erich Kaltofen. Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation*, 29(6):891–919, 2000.

[30] M. Kerber, M. Kohlhase, and V. Sorge. An integration of mechanised reasoning and computer algebra that respects explicit proofs. Technical Report CSRP-96-9, University of Birmingham, 1996.

[31] M. Kohlhase. OMDoc: An open markup format for mathematical documents (version 1.1). Technical report, Carnegie Mellon University, 2002.

[32] Marni Mishna. Attribute grammars and automatic complexity analysis. *Advances in Applied Mathematics*, 30:189–207, 2003.

[33] R. Risch. The problem of integration in finite terms. *Trans. Amer. Math. Soc.*, 139:167–189, 1969.

[34] Y. V. Srinivas and R. Jullig. Specware: Formal support for composing software. In *Mathematics of Program Construction*, pages 399–422, 1995.

[35] D.R. Stoutemeyer. Crimes and misdemeanors in the computer algbebra trade. *Notices of the AMS*, pages 701–785, 1991.

[36] L. Van Den Dries. Alfred Tarski's elimination theory for real closed fields. *Journal of Symbolic Logic*, 53:7–19, 1988.

[37] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Series in Computer Science. Prentice Hall, 1996.