Composing Hidden Information Modules over Inclusive Institutions

Joseph Goguen
Department of Computer Science & Engineering
University of California at San Diego
E-mail: goguen@cs.ucsd.edu

Grigore Roşu
Department of Computer Science
University of Illinois at Urbana-Champaign
E-mail: grosu@cs.uiuc.edu

Abstract: This paper studies the composition of modules that can hide information, over a very general class of logical systems called inclusive institutions. Two semantics are given for composition of such modules using five familiar operations, and a property called conservativity is shown necessary and sufficient for these semantics to agree. The first semantics extracts the visible properties of the result of composing the visible and hidden parts of modules, while the second uses only the visible properties of the components; the semantics agree when the visible consequences of hidden information are enough to determine the result of the composition. A number of "laws of software composition" are proved relating the composition operations. Inclusive institutions simplify many proofs.

1 Introduction

Modularization limits the complexity of large systems by composing them from parts; this eases both initial construction and later modification, and also facilitates reuse. Parameterized programming [10, 11] significantly further enhances flexibility and reusability, by providing *parameterized modules* along with *views*, also called *fitting morphisms*, which say how to fit the syntax of a formal parameter to an actual parameter in a convenient, flexible way, including defaults when there is only one choice; moreover, views can be parameterized, dependent types are supported through formal parameters that are parameterized by previously introduced formal parameters, and *module expressions* compose modules into systems. The module composition operations in this paper are for aggregating, renaming, enriching, hiding, and instantiating parameterized modules. *Module expressions* are terms built from basic modules, parameterized modules, and views, using these five operations. We believe views are key to realizing the full practical potential of modularization. Note that the results of this paper are by no means limited to specification languages, let alone equational languages, and indeed, a slightly more concrete version has been used to greatly extend Ada generics [26, 16].

There are several good reasons to hide information in modules. First, as emphasized by Parnas [21], information hiding supports data abstraction, and more generally, allows replacing one module by another having the same semantics for its visible signature, but a different implementation, without having to worry that other modules might have used details of the implementation. Second, a classic result of Bergstra and Tucker [2] says that every computable algebra has a finite equational specification with some hidden operations, and examples show that the hidden operations are sometimes necessary (see [19] for a survey of this area). Third, [14] shows that every [finite] behavioral (also called observational, or hidden) algebraic specification [13, 24, 14, 22] has an equivalent [finite] information hiding specification with the same models, but using ordinary satisfaction.

Category theory and institutions are heavily used in this paper. Institutions (see Section 2.3) formalize the informal notion of logical system, with a balanced interplay of syntax and semantics, to support research that is independent of the underlying logic. An institution consists of: a category of signatures; a functor from signatures Σ to classes of Σ-sentences; a contravariant functor from signatures Σ to categories of Σ-models; and for every signature Σ, a relation of satisfaction $\models_\Sigma$ between Σ-sentences and Σ-models, such that $M' \models_{\Sigma'} \varphi(f)$ iff $\varphi(M') \models_\Sigma f$, for every signature morphism $\varphi\colon \Sigma \to \Sigma'$, every Σ'-model M', and every Σ-sentence f. Given a class A of Σ-sentences, let A^* denote the class of Σ-models that satisfy every sentence in A, and given a class V of Σ-models, let V^* denote the class of Σ-sentences that are satisfied by all models in V. Modularization has been one of the most important applications of institutions; other topics include borrowing of logics, and translations between logics. Many logical systems have been shown to be institutions, and most recent algebraic specification languages have an institution-based semantics. A recent summary of results appears in [15].

Defining and proving properties of module systems can be greatly simplified when the institution involved is *inclusive*, in the sense that its category of signatures satisfies certain natural conditions that axiomatize the notion of inclusion. (Our inclusive institutions are simpler than the original ones in [9] because we use the inclusive categories of [15]; see Section 2.2.) It seems that all institutions proposed for specification or programming are inclusive. Several properties of inclusive institutions are proved, including a generalization of the Closure Lemma of classical institution theory.

Let $\mathcal{I}$ be a fixed inclusive institution. In this paper, we let the term *module* refer to a triple (Φ, Σ, A) where Φ is a subsignature of Σ, and A is a set of Σ-sentences, all from $\mathcal{I}$. Σ is called the *working signature*, and it includes both the public and the private operations of the module; Φ is called the *visible signature*, which defines the public interface; elements of A are called *axioms*. The *visible theorems* of $M = (\Phi, \Sigma, A)$ are the Φ-sentences in the "double star closure" A^{**} of A, also denoted $Vth(M)$. A *model* of M is a Φ-model of its visible theorems. A *transparent module* has $\Phi = \Sigma$; these correspond to traditional specifications without information hiding.

In the first semantics, the meaning of a module expression is the visible theory of the result of evaluating the module expression compositionally (i.e., recursively) over the five operations. For example, if $M = (\Phi, \Sigma, A)$ then $[\![M]\!]_1 = Vth(M)$, and given also $M' = (\Phi', \Sigma', A')$, then $[\![M + M']\!]_1 = Vth(M + M')$, where $M + M'$ is defined to be $(\Phi \cup \Phi', \Sigma \cup \Sigma', A \cup A')$. This semantics is the same as that in [16], except that signature union comes from inclusive categories rather than the extended set theory used in [16].

In our second semantics, from [23], meaning is directly compositional over visible theories. As before, $[\![M]\!]_2 = Vth(M)$, but now $[\![M + M']\!]_2 = ([\![M]\!]_2 \cup [\![M']\!]_2)^{**}$, where this closure is relative to the signature $\Phi \cup \Phi'$. Meanings for the other module composition operations are similar, and do not use the hidden parts of component modules.

The first semantics is more comprehensive because it uses more information; it is a non-trivial theorem that the two semantics agree when all modules involved are conservative, where a module (Φ, Σ, A) is *conservative* iff every Φ-model of its visible theorems can be extended to a Σ-model of A. Transparent modules are obviously conservative, but in general, testing for conservativity depends on the institution involved, and can be difficult. The modules that arise in practice for equational institutions are conservative. One approach for showing conservativity in equational institutions is to show that every Φ-algebra can be enriched with private carriers and operations, such that it satisfies the axioms. Since each semantics gives a theory, each has an associated class of models, and these two classes also agree under conservativity. An example in the equational institution showing that conservativity is necessary as well as sufficient is given at the end of Section 5.1.

Other results in this paper include a number of identities that hold among the meanings of simple module expressions. These can be considered "basic laws of software engineering" (though they are mostly very simple); such laws were used in the LILEANNA system to simplify module expressions before handing them over to the backend of the Ada compiler for optimization [26, 16]. For example, if we write $E \equiv E'$ to indicate that $[\![E]\!]_1 = [\![E']\!]_1$ for module expressions E, E', then we have $E + E' \equiv E' + E$, and also $E + (E' + E'') \equiv (E + E') + E''$. Of course, these are only the simplest examples; the most interesting identities involve the instantiation of parameterized modules.

1.1 Related Work

We discuss only work that directly influenced this paper; readers wanting more background and historical information should consult references in the cited papers. The most influential were [16] and [9], by Goguen and Tracz, and by Diaconescu, Goguen and Stefaneas, respectively. The first considers modules with hidden parts, using a less abstract version of institutions, in which signatures are structured set/tuple hierarchies, so that inclusions are directly available and need not be axiomatized. This framework is less general than that of the present paper[1], but it has the advantage of making the techniques for implementing our module system more explicit. On the other hand, proofs in this framework are less elegant, and more difficult to discover. LILEANNA [26, 16] actually implements parameterized programming with modules that can hide information, using Ada as its programming languages and Anna as its specification language. Modularization over inclusive institutions in studied in [9], but its modules do not hide information, and its notion of inclusion system is less general then the present one. We paper seems a natural next step in research on parameterized programming, as begun in [10, 11], inspired by the Clear module system [3], and further developed in [16] and [9].

Bergstra, Heering and Klint [1] axiomatize several operations on modules and prove certain properties, including a normalization theorem; unfortunately, first order logic is built into their formalism for sentences, which limits the application of their results. When our institution is first order logic, then results in this paper prove from more basic principles all of the axioms in [1] that concern our operations.

Inclusive institutions seem an attractive alternative to Mossakowski's "institutions with symbols" [20], which assign sets of symbols to signatures, because inclusions automatically keep track of shared symbols in subsignatures, while allowing the usual operations on modules to be easily and naturally expressed. Mossakowski's approach was developed for the semantics of the European CASL specification language [4].

Diaconescu [7] studies modularization for category-based equational logic (CBEL). which is limited to equational-like logics, and does not consider information hiding. We believe that under some mild conditions, CBEL is an inclusive institution, so that results in this paper would apply to it.

Section 2 gives notation and concepts from category theory, inclusive categories and institutions. Section 3 presents inclusive institutions, and Section 4 introduces modules. Section 5 is the main section of the paper, giving the five module operations with their semantics and the basic laws. Many of the proofs omitted here can be found in [23]; the proof of Theorem 16 is included because of its importance and elegance.

2 Preliminaries

Categories, inclusions and institutions are heavily used in this paper, and this section briefly introduces our notation and terminology for these concepts.

[1]Although it seems to include all the standard examples.

2.1 Category Theory

The reader is assumed to be familiar with basics of category theory, including limits, colimits, functors, and adjoints [18, 17]. $|\mathcal{C}|$ denotes the class of objects of a category $\mathcal{C}$, and $\mathcal{C}(A, B)$ denotes the set of morphisms in $\mathcal{C}$ from object A to object B. The composition of morphisms is written in diagrammatic order, that is, $f; g\colon A \to C$ is the composition of $f\colon A \to B$ with $g\colon B \to C$. **Cat** denotes the category with small categories as objects and functors as morphisms. A family of morphisms $\{e_i\colon A_i \to B \mid i \in I\}$ is **epimorphic** iff for any two morphisms $f, g\colon B \to C$, if $e_i; f = e_i; g$ for each $i \in I$ then $f = g$.

A functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{D}$ is **full** (**faithful**) if its restrictions $\mathcal{F}\colon \mathcal{C}(A, B) \to \mathcal{D}(\mathcal{F}(A), \mathcal{F}(B))$ are surjective (injective) for all objects A, B in $\mathcal{C}$. $\mathcal{F}$ is **dense** provided that for each $D \in |\mathcal{D}|$ there is some $C \in |\mathcal{C}|$ such that $\mathcal{F}(C)$ is isomorphic to D. A **full subcategory** is a subcategory such that the inclusion functor is full. A category is **skeletal** iff isomorphic objects are identical. A **skeleton** of a category $\mathcal{C}$ is a maximal full skeletal subcategory of $\mathcal{C}$; it can be shown that any two skeletons of a category are isomorphic in **Cat**. A category $\mathcal{C}$ is **equivalent** to a category $\mathcal{D}$ iff C and $\mathcal{D}$ have isomorphic skeletons. It is known [18] that two categories $\mathcal{C}$ and $\mathcal{D}$ are equivalent iff there exists a functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{D}$ which is full, faithful and dense.

Pullbacks in **Cat** have the following special property: if a pair of functors $\mathcal{F}_1\colon \mathcal{P} \to \mathcal{C}_1$ and $\mathcal{F}_2\colon \mathcal{P} \to \mathcal{C}_2$ is a pullback in **Cat** of $\mathcal{G}_1\colon \mathcal{C}_1 \to \mathcal{D}$ and $\mathcal{G}_2\colon \mathcal{C}_2 \to \mathcal{D}$, and if $C_1 \in |\mathcal{C}_1|$ and $C_2 \in |\mathcal{C}_2|$ are such that $\mathcal{G}_1(C_1) = \mathcal{G}_2(C_2)$, then there is a *unique* object P in $\mathcal{P}$ such that $\mathcal{F}_1(P) = C_1$ and $\mathcal{F}_2(P) = C_2$.

2.2 Inclusive Categories

Many categories have certain morphisms which are intuitively inclusions. The problem of characterizing such morphisms was raised in [12], first solved in [9] with the notion of inclusion system, and further developed and simplified in [5, 6, 23, 15]. The simplest version occurs in [15]: An **inclusive category** $\mathcal{C}$ is a category having a broad subcategory[2] $\mathcal{I}$ which is a poclass (i.e., its objects are a class such that $\mathcal{I}(A, B)$ has at most one element, and if both $\mathcal{I}(A, B)$ and $\mathcal{I}(B, A)$ are non-empty, then $A = B$) with finite products and coproducts, called **intersection** (denoted $\cap$) and **union** (denoted $\cup$ or possibly $+$), respectively, such that for every pair A, B of objects, $A \cup B$ is a pushout of $A \cap B$ in $\mathcal{C}$; morphisms in $\mathcal{I}$ are written $\hookrightarrow$ and called **inclusions**. In particular, $\mathcal{C}$ and $\mathcal{I}$ have an initial object, which we denote $\emptyset$, and $A_1, ..., A_n$ are **disjoint** iff $A_i \cap A_j = \emptyset$ for $i \neq j$. $\mathcal{C}$ is **distributive** iff $\mathcal{I}$ is distributive, in the sense that $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ (which, as is lattice theory, is equivalent to $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$) in $\mathcal{I}$.

Proposition 1 *In any category $\mathcal{C}$ with strong inclusions $\mathcal{I}$:*

1. *The family of inclusions $A_i \hookrightarrow \bigcup_{j=1}^{n} A_j$ for $i = 1, ..., n$ is epimorphic.*

[2]In the sense that it has the same objects as $\mathcal{C}$.

2. *$\bigcup_{j=1}^{n} A_j$ is a colimit in $\mathcal{C}$ of the diagram given by the pairs of inclusions $A_i \cap A_j \hookrightarrow A_i$ and $A_i \cap A_j \hookrightarrow A_j$ for $i, j = 1, ..., n$.*

We say that morphisms $h_i\colon A_i \to B_i$ in $\mathcal{C}$ for $i = 1, ..., n$ **have a union**, written $\bigcup_{j=1}^{n} h_j$, iff there is a morphism $h\colon \bigcup_{j=1}^{n} A_j \to \bigcup_{j=1}^{n} B_j$ such that $(A_i \hookrightarrow \bigcup_{j=1}^{n} A_j); h = h_i; (B_i \hookrightarrow \bigcup_{j=1}^{n} B_j)$ for $i = 1, ..., n$. Such a morphism h is unique if it exists, and $h_1, ..., h_n$ have a union whenever $A_1, ..., A_n$ are disjoint. $\mathcal{C}$ **has pushouts which preserve inclusions** iff for any pair of arrows $(A \hookrightarrow B, A \to A')$ there are pushouts of the form $(A' \hookrightarrow B', B \to B')$. A functor between two inclusive categories is **inclusive** (or **preserves inclusions**) iff it takes inclusions in the source category to inclusions in the target category.

2.3 Institutions

Institutions were introduced by Goguen and Burstall [12] to formalize the intuitive notion of logical system. An **institution** consists of a category **Sign** whose objects are called **signatures**, a functor $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$ giving for each signature a set whose elements are called **Σ-sentences**, a functor $\mathbf{Mod}\colon \mathbf{Sign} \to \mathbf{Cat}^{op}$ giving for each signature Σ a category of **Σ-models**, and a signature-indexed relation called **satisfaction**, $\models\, = \{\models_\Sigma \mid \Sigma \in \mathbf{Sign}\}$, where $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$, such that for each signature morphism $h\colon \Sigma \to \Sigma'$, the **satisfaction condition**, $m' \models_{\Sigma'} \mathbf{Sen}(h)(a)$ iff $\mathbf{Mod}(h)(m') \models_\Sigma a$, holds for all $m' \in |\mathbf{Mod}(\Sigma')|$ and $a \in \mathbf{Sen}(\Sigma)$. We may write h instead of $\mathbf{Sen}(h)$ and $\_\restriction_h$ instead of $\mathbf{Mod}(h)$; $m\restriction_h$ is called the **h-reduct** of m. The satisfaction condition then takes the simpler form $m' \models_{\Sigma'} h(a)$ iff $m'\restriction_h \models_\Sigma a$. We may omit the subscript Σ when it can be inferred from context. Given a set A of Σ-sentences, let

$$A^* = \{m \in \mathbf{Mod}(\Sigma) \mid m \models_\Sigma a \text{ for all } a \in A\}\ ,$$

and given a class V of Σ-models, let

$$V^* = \{a \in \mathbf{Sen}(\Sigma) \mid m \models_\Sigma a \text{ for all } m \in V\}\ .$$

Then the **closure** of a set of Σ-sentences A is $A^\bullet = A^{**}$, and $\_^\bullet$ is a closure operator, i.e., it is extensive, monotonic and idempotent; the sentences in $A^\bullet$ are called the **theorems of** A.

A **specification** or **presentation** is a pair (Σ, A) where Σ is a signature and A is a set of Σ-sentences. A **specification morphism** from (Σ, A) to (Σ', A') is a signature morphism $h\colon \Sigma \to \Sigma'$ such that $h(A) \subseteq A'^\bullet$. Specifications and specification morphisms give a category denoted **Spec**. A **theory** (Σ, A) is a specification with $A = A^\bullet$; the full subcategory of theories in **Spec** is denoted **Th**. Given a specification (Σ, A), $\mathbf{Mod}(\Sigma, A)$ denotes the full subcategory of $\mathbf{Mod}(\Sigma)$ of models that satisfy A; given a morphism $h\colon (\Sigma, A) \to (\Sigma', A')$, $\_\restriction_h$ takes models of A' to models of A. We will also use $\models_\Sigma$ for sets of sentences: $A \models_\Sigma B$ iff $\mathbf{Mod}(\Sigma, A) \subseteq \mathbf{Mod}(\Sigma, B)$. **Th** and **Spec** are equivalent categories, where the equivalence functor is just the inclusion $\mathcal{U}_s\colon \mathbf{Th} \to \mathbf{Spec}$. This functor has a left-adjoint-left-inverse $\mathcal{F}_s : \mathbf{Spec} \to \mathbf{Th}$, given by $\mathcal{F}_s(\Sigma, A) =$

$(\Sigma, A^\bullet)$ on objects and the identity on morphisms; note that $\mathcal{F}_s$ is also right adjoint to $\mathcal{U}_s$, so that **Th** is a reflective and coreflective subcategory of **Spec**.

A theory morphism $h\colon (\Sigma, A) \to (\Sigma', A')$ is **conservative** iff for any (Σ, A)-model m there is a (Σ', A')-model m' such that $m'\restriction_h = m$, i.e., iff its retract map $\_\restriction_h\colon \mathbf{Mod}(\Sigma', A') \to \mathbf{Mod}(\Sigma, A)$ is surjective. A signature morphism $h\colon \Sigma \to \Sigma'$ is **conservative** iff it is conservative as a morphism of **void** theories, i.e., iff $h\colon (\Sigma, \emptyset^\bullet) \to (\Sigma', \emptyset'^\bullet)$ is conservative. An important result of [12] is that **Th** is cocomplete whenever **Sign** is cocomplete; therefore **Th** has pushouts whenever **Sign** has pushouts: if $h_1\colon (\Sigma, A) \to (\Sigma_1, A_1)$ and $h_2\colon (\Sigma, A) \to (\Sigma_2, A_2)$ are theory morphisms, and if $(h'_1\colon \Sigma_1 \to \Sigma',\ h'_2\colon \Sigma_2 \to \Sigma')$ is a pushout of (h_1, h_2) in **Sign**, then $(h'_1\colon (\Sigma_1, A_1) \to (\Sigma', A'), h'_2\colon (\Sigma_2, A_2) \to (\Sigma', A'))$ is a pushout of h_1, h_2 in **Th**, where $A' = (h'_1(A_1) \cup h'_2(A_2))^\bullet$.

Proposition 2 *Given $h\colon \Sigma \to \Sigma'$, $A, A' \subseteq \mathbf{Sen}(\Sigma)$, and $a \in \mathbf{Sen}(\Sigma)$, then:*

1. *<u>Closure Lemma:</u> $h(A^\bullet) \subseteq h(A)^\bullet$, i.e., $A \models_\Sigma a$ implies $h(A) \models_{\Sigma'} h(a)$.*
2. *If h is conservative, then $A \models_\Sigma a$ iff $h(A) \models_{\Sigma'} h(a)$.*
3. *$h(A^\bullet)^\bullet = h(A)^\bullet$.*
4. *$(A^\bullet \cup A')^\bullet = (A \cup A')^\bullet$.*

For example, the institution for LILEANNA [26, 16] has Anna as its specification language, and Ada programs as its models.

3 Inclusive Institutions

The semantics of module systems over an institution is much simplified when signature inclusions are available, in the following sense:

Definition 3 An institution is **inclusive** iff **Sign** is inclusive and has pushouts which preserve inclusions, **Sen** is inclusive, and **Mod** preserves pushouts which preserve inclusions, i.e., takes them to pullbacks in **Cat**. An inclusive institution is **distributive** iff its category of signatures is distributive. ■

We now fix an inclusive institution and refer to it as the "given institution." Many natural properties can be expressed intuitively in inclusive institutions, for example, if A and A' are sets of Σ- and Σ'-sentences, respectively, then $(A^\bullet \cup A'^\bullet)^\bullet = (A \cup A')^\bullet$, where the outermost closures are done over $\Sigma \cup \Sigma'$-sentences; also, if $\Phi \hookrightarrow \Sigma$, $a \in \mathbf{Sen}(\Phi)$, and $A \subseteq \mathbf{Sen}(\Phi)$, then $A \models_\Phi a$ implies $A \models_\Sigma a$, with equivalence when $\Phi \hookrightarrow \Sigma$ is conservative. The category **Th** tends to have many of the properties of **Sign**. In particular,

Proposition 4 *For any inclusive institution,* **Th** *is inclusive and has pushouts that preserve inclusions.*

Proof: The category of inclusions $\mathcal{I}_{\mathbf{Th}}$ in **Th** consists of morphisms $(\Sigma, A) \hookrightarrow (\Sigma', A')$ where $\Sigma \hookrightarrow \Sigma'$ is an inclusion in **Sign** and $A \subseteq A'$. It is easy to check that $\mathcal{I}_{\mathbf{Th}}$ is a poclass with the same objects as **Th**. Define the union of theories (Σ, A), (Σ', A') by $(\Sigma, A) \cup (\Sigma', A') = (\Sigma \cup \Sigma', (A \cup A')^\bullet)$ where the closure is over $(\Sigma \cup \Sigma')$-sentences, and define their intersection by $(\Sigma, A) \cap (\Sigma', A') = (\Sigma \cap \Sigma', A \cap A')$. We now show correctness of these definitions. That union is a pushout of an intersection in $\mathcal{I}_{\mathbf{Th}}$ follows from the construction of pushouts in **Th**. Now consider an inclusion $(\Sigma, A) \hookrightarrow (\Sigma_1, A_1)$ in **Th** and a morphism $h\colon (\Sigma, A) \to (\Sigma_2, A_2)$ in **Th**. Let $(\Sigma_2 \hookrightarrow \Sigma', h_{\Sigma'}\colon \Sigma_1 \to \Sigma')$ be a pushout of $(\Sigma \hookrightarrow \Sigma_1, h\colon \Sigma \to \Sigma_2)$ in **Sign** which preserves the inclusion. Then $((\Sigma_2, A_2) \hookrightarrow (\Sigma', A'), h_{\Sigma'}\colon (\Sigma_1, A_1) \to (\Sigma', A'))$ is the desired pushout in **Th**, where $A' = (A_2 \cup h_{\Sigma'}(A_1))^\bullet$, again by the construction of pushouts in **Th**. □

Convention 5 We do not assume any particular way to calculate pushouts of signatures, nor do we require these pushouts to have any special properties, but for notational convenience, we assume a *fixed* pushout that preserves the inclusions of diagrams $(\Phi \hookrightarrow \Sigma,\ h\colon \Phi \to \Phi')$; let $(\Phi' \hookrightarrow \Sigma_h,\ h_\Sigma\colon \Sigma \to \Sigma_h)$ denote this pushout. Also, we may say **theory extension** instead of theory inclusion. ■

Open Problem It would be useful to have an algorithm for pushouts of the usual signatures, that is closed under horizontal and/or vertical composition, i.e., such that $\Sigma'_{(h_\Sigma)} = \Sigma'_h$ for any $(\Phi \hookrightarrow \Sigma,\ h\colon \Phi \to \Phi')$ and $\Sigma \hookrightarrow \Sigma'$, and such that $(\Sigma_h)_g = \Sigma_{h;g}$ for any signature morphism g with source Φ'.

Definition 6 Given $\imath\colon \Phi \hookrightarrow \Sigma$ in **Sign** and $A \subseteq \mathbf{Sen}(\Sigma)$, let $Th^\Sigma_\Phi(A)$ denote $\imath^{-1}(A^\bullet) \subseteq \mathbf{Sen}(\Phi)$, called the **$\Phi$-visible theorems of** A (over Σ). ■

$Th^\Sigma_\Phi(A)$ contains all the Φ-sentences that are consequences of A. When $\imath$ is an identity, then $Th^\Sigma_\Sigma(A) = A^\bullet$.

Example 7 Assume a logic where equational reasoning and induction are sound, and let *LIST* be a specification of lists containing at least the sorts *Elt* and *List*, a constant *nil* of sort *List*, and a constructor $cons\colon Elt \times List \to List$. Let Φ extend the signature of lists by a reverse operation $rev\colon List \to List$, let Σ extend Φ by a private operation $aux\colon List \times List \to List$, and let A contain the equations

- $(\forall L : List)\ rev(L) = aux(L, nil)$.
- $(\forall P : List)\ aux(nil, P) = P$.
- $(\forall E : Elt; L, P : List)\ aux(cons(E, L), P) = aux(L, cons(E, P))$.

Then the following are two Φ-visible theorems of A over Σ:

- $rev(nil) = nil$,
- $(\forall L : List)\ rev(rev(L)) = L$.

The proof of the second requires induction and two lemmas. ■

The following properties are familiar for many particular logics, because they hold in any inclusive institution:

Proposition 8 *If $\Psi \hookrightarrow \Phi \hookrightarrow \Sigma$, $A \subseteq A' \subseteq \mathbf{Sen}(\Sigma)$, and $B \subseteq \mathbf{Sen}(\Phi)$, then:*

1. *$B \subseteq Th^{\Sigma}_{\Phi}(B)$.*
2. *$Th^{\Phi}_{\Psi}(B) \subseteq Th^{\Sigma}_{\Psi}(B)$.*
3. *$Th^{\Phi}_{\Psi}(B) = Th^{\Sigma}_{\Psi}(B)$ if $\Phi \hookrightarrow \Sigma$ is conservative.*
4. *$Th^{\Sigma}_{\Psi}(A) \subseteq Th^{\Sigma}_{\Phi}(A)$.*
5. *$Th^{\Sigma}_{\Phi}(A) \subseteq Th^{\Sigma}_{\Phi}(A')$.*
6. *$Th^{\Sigma}_{\Psi}(A) \subseteq Th^{\Sigma}_{\Phi}(Th^{\Sigma}_{\Psi}(A))$.*
7. *$Th^{\Sigma}_{\Psi}(Th^{\Sigma}_{\Phi}(A)) \subseteq Th^{\Sigma}_{\Psi}(A)$.*
8. *$Th^{\Sigma}_{\Phi}(Th^{\Sigma}_{\Phi}(A)) = Th^{\Sigma}_{\Phi}(A)$.*
9. *$Th^{\Phi}_{\Phi}(Th^{\Sigma}_{\Phi}(A)) = Th^{\Sigma}_{\Phi}(A)$.*
10. *$Th^{\Phi}_{\Psi}(Th^{\Sigma}_{\Phi}(A)) = Th^{\Sigma}_{\Psi}(A)$.*

Proof: Let $\imath' : \Psi \to \Phi$ and $\imath : \Phi \to \Sigma$ be the two inclusions.

1. If $b \in B$ then $B \models_{\Sigma} b$, i.e., $b \in Th^{\Sigma}_{\Phi}(B)$.
2. By 1. of Proposition 2.
3. This is exactly 2. in Proposition 2.
4. Since **Sen** is a morphism of inclusion systems, a is in $\mathbf{Sen}(\Phi)$ whenever a is in $\mathbf{Sen}(\Psi)$.
5. This is equivalent to $\imath^{-1}(A^{\bullet}) \subseteq \imath^{-1}(A'^{\bullet})$, which holds because $A^{\bullet} \subseteq A'^{\bullet}$.
6. This follows from 1. with $Th^{\Sigma}_{\Psi}(A)$ for B.
7. This is equivalent to $(\imath';\imath)^{-1}(\imath^{-1}(A^{\bullet})) \subseteq (\imath';\imath)^{-1}(A^{\bullet})$, which is true because $\imath^{-1}(A^{\bullet}) \subseteq A^{\bullet}$.
8. This follows from 6. and 7., with $\Psi = \Phi$.
9. By 1., $Th^{\Sigma}_{\Phi}(A) \subseteq Th^{\Phi}_{\Phi}(Th^{\Sigma}_{\Phi}(A))$. On the other hand, $Th^{\Phi}_{\Phi}(Th^{\Sigma}_{\Phi}(A)) \subseteq Th^{\Sigma}_{\Phi}(Th^{\Sigma}_{\Phi}(A))$ by 2., and also $Th^{\Phi}_{\Phi}(Th^{\Sigma}_{\Phi}(A)) \subseteq Th^{\Sigma}_{\Phi}(A)$ by 8.
10. This is equivalent to $\imath'^{-1}(\imath^{-1}(A^{\bullet})) = (\imath';\imath)^{-1}(A^{\bullet})$, which is true.

□

Lemma 9 <u>**Generalized Closure Lemma:**</u> *Given $A \subseteq \mathbf{Sen}(\Sigma)$, inclusions $\imath : \Phi \hookrightarrow \Sigma$, $\imath' : \Phi' \hookrightarrow \Sigma'$, and morphisms $h : \Phi \to \Phi'$ and $g : \Sigma \to \Sigma'$ such that the diagram*

$$\begin{tikzcd}
\Phi \arrow[r, hook, "\imath"] \arrow[d, "h"'] & \Sigma \arrow[d, "g"] \\
\Phi' \arrow[r, hook, "\imath'"'] & \Sigma'
\end{tikzcd}$$

commutes, then $h(Th^{\Sigma}_{\Phi}(A)) \subseteq Th^{\Sigma'}_{\Phi'}(g(A))$.

Proof: Let a be a Φ-sentence in $Th^{\Sigma}_{\Phi}(A)$, so that $A \models_{\Sigma} a$. Then $g(A) \models_{\Sigma'} g(a)$ by the classic Closure Lemma (1. of Proposition 2). But $g(a) = h(a)$ because **Sen** preserves inclusions, and so $h(a)$ is in $Th^{\Sigma'}_{\Phi'}(g(A))$. $\Box$

The classic Closure Lemma is the special case of the above where $\imath, \imath'$ are identities, that is, where there are no private features.

4 Modules

Our modules, like those in [16], extend the usual algebraic specifications by allowing hidden information, which may be used in defining visible features.

Definition 10 A **module** is a triple (Φ, Σ, A), where $\Phi \hookrightarrow \Sigma$ are signatures and A is a set of Σ-sentences; Φ is called the **visible signature**, Σ the **working signature**, $Th(M) = Th^{\Sigma}_{\Sigma}(A)$ the **working theorems**, and $Vth(M) = Th^{\Sigma}_{\Phi}(A)$ the **visible theorems**. A **morphism** $h\colon M \to M'$ of modules is a morphism of their visible signatures such that $h(Vth(M)) \subseteq Vth(M')$. ■

Modules together with module morphisms form a category **MSpec**. The functor $\mathcal{M}$ from **Th** to **MSpec** defined by $\mathcal{M}(\Sigma, A) = (\Sigma, \Sigma, A)$ and $\mathcal{M}(h) = h$, is full, faithful and dense, i.e., is an equivalence of categories. Further (by Theorem 1, page 91 of [18]), $\mathcal{M}$ is part of an adjoint equivalence, with left adjoint $\mathcal{T}\colon$ **MSpec** $\to$ **Th** defined by $\mathcal{T}(\Phi, \Sigma, A) = (\Phi, Th^{\Sigma}_{\Phi}(A))$ on objects, and the identity on morphisms; the unit of this adjunction is $1_{\Phi}\colon (\Phi, \Sigma, A) \to (\Phi, \Phi, Th^{\Sigma}_{\Phi}(A))$. We let $\mathcal{U}\colon$ **MSpec** $\to$ **MSpec** denote the functor $\mathcal{T};\mathcal{M}$, taking modules (Φ, Σ, A) to modules $(\Phi, \Phi, Th^{\Sigma}_{\Phi}(A))$. Notice that $\mathcal{T}$ is also a right adjoint of $\mathcal{M}$, so that **Th** is (modulo isomorphism) a reflective and coreflective subcategory of **MSpec**. Since the two categories are equivalent, every categorical property [17] of **Th** is also a property of **MSpec**. In particular, pushouts are preserved and reflected by $\mathcal{M}$ and $\mathcal{T}$, and **MSpec** is cocomplete whenever **Sign** is cocomplete, since **Th** is cocomplete (by [12]).

Definition 11 A Φ-model m **satisfies** $M = (\Phi, \Sigma, A)$ iff $m \models_{\Phi} Vth(M)$; in this case, we write $m \models M$. ■

If $h\colon M \to M'$ is a module morphism and $m' \models M'$, then $m'\restriction_h \models M$. Therefore in any inclusive institution, the functor **Mod** extends to **MSpec** by mapping a module M to the full subcategory **Mod**(M) of **Mod**(Φ) with the Φ-models satisfying M as its objects.

It is common to call a theory extension $(\Sigma, A) \hookrightarrow (\Sigma, A')$ conservative when $A = A' \cap \mathbf{Sen}(\Sigma)$; we call this notion **syntactic conservativity** to distinguish it from the semantic version. Notice that for any module $M = (\Phi, \Sigma, A)$, the theory inclusion $(\Phi, Vth(M)) \hookrightarrow (\Sigma, Th(M))$ is syntactically conservative, because $Vth(M) = Th^{\Sigma}_{\Phi}(A) = \{a \in \mathbf{Sen}(\Phi) \mid A \models_{\Sigma} a\} = \{a \in \mathbf{Sen}(\Phi) \mid a \in Th(M)\} = Th(M) \cap \mathbf{Sen}(\Phi)$. As shown in [9], syntactic conservativity is a necessary but insufficient condition for semantic conservativity. So it is not surprising that we also need a stronger conservativity for modules:

Definition 12 A module M is **conservative** if and only if the theory inclusion $(\Phi,\ \mathit{Vth}(M)) \hookrightarrow (\Sigma,\ \mathit{Th}(M))$ is conservative. ■

Transparent modules, with $\Phi = \Sigma$, are always conservative. But there are simple non-conservative modules, even for unsorted equational logic, such as the following (after [9]):

Example 13 Let Φ contain a constant a and a unary operation $-$, let Σ additionally contain a constant c, and let A contain the equation $c = -c$. Then the visible theorems of this module form an empty theory, but there are Φ-models that cannot be extended to Σ-models satisfying A, such as $m = \{1, -1\}$ with $m_a = 1$, $m_-(1) = -1$ and $m_-(-1) = 1$. ■

5 Operations on Modules

We give two natural semantics for each module combining operation. The first (from [16]) takes the visible theorems of the combined module, while the second (from [23]) combines the visible theorems of the component modules. The module composition used in the first semantics essentially combines the parts of the component modules, with renamings to avoid name conflicts, while the second semantics combines their visible theorems directly. The first semantics is more comprehensive because it makes use of more information, and the two can agree only under special conditions, when interactions among hidden information can be safely ignored. Note that when proving theorems, it may be easier to use private information than to use only its consequences; for example, the former may be finite while the latter is infinite. We show that conservativity is a necessary and sufficient condition for the two semantics to agree; this result can be seen as saying when it is safe to ignore interactions among hidden parts. All five module operations have simple definitions over any inclusive institution, corresponding to natural ways to implement them, and they all preserve conservativity under natural conditions.

5.1 Aggregation

The aggregation of modules is essentially the union of their various parts. However, this simple view is complicated by the need to handle symbols having the same name but defined in different modules, and symbols coming from shared submodules. A standard way to prevent name clashes is to tag symbols with the name of the module where they are defined; symbols defined in shared submodules are then tagged just once. Some languages with overloaded operators have complex symbol resolution algorithms, while others take the simple union of all symbols, leaving the name collision problem to the user. The latter approach is actually appropriate for our purpose, which is to investigate the role of private symbols in the semantics of aggregation. Moreover, if symbols are already tagged with their originating module when they are declared, with their untagged name available as a convenient abbreviation when it is unambiguous (as is done in OBJ), this approach is equivalent to the standard one.

We will show that the two semantics for aggregation agree when the component modules are conservative and all the symbols that they share are visible, and we will give counterexamples showing that both these requirements are necessary. We also give some simple "laws of software composition" and prove a number of other basic properties of aggregation.

Definition 14 Given modules $M = (\Phi, \Sigma, A)$ and $M' = (\Phi', \Sigma', A')$, their **aggregation** $M + M'$ is $(\Phi \cup \Phi', \Sigma \cup \Sigma', A \cup A')$. We then let $[\![M + M']\!]_1 = \mathit{Vth}(\Phi \cup \Phi', \Sigma \cup \Sigma', A \cup A')$, and $[\![M + M']\!]_2 = (\mathit{Vth}(M) \cup \mathit{Vth}(M'))^\bullet$, where the closure is with respect to $\Phi \cup \Phi'$. ■

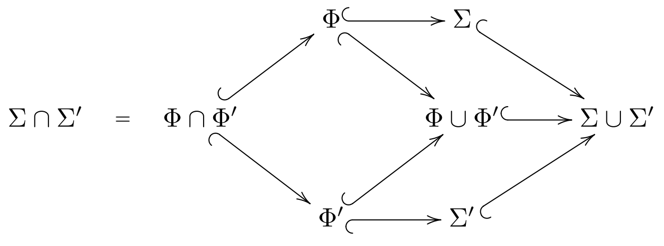
This definition makes sense because $\Phi \cup \Phi' \hookrightarrow \Sigma \cup \Sigma'$ and $A \cup A'$ is a set of $(\Sigma \cup \Sigma')$-sentences, since **Sen** preserves inclusions.

Proposition 15 *Aggregation is commutative, associative and idempotent because union is. If $\imath$ and $\imath'$ denote the inclusions $\Phi \hookrightarrow \Phi \cup \Phi'$ and $\Phi' \hookrightarrow \Phi \cup \Phi'$, respectively, then $\imath\colon M \to M + M'$ and $\imath'\colon M' \to M + M'$ are module morphisms. Therefore $m \models M + M'$ implies $m\restriction_\Phi \models M$ and $m\restriction_{\Phi'} \models M'$ for any $(\Phi \cup \Phi')$-model m.*

We call the following a "theorem" because of its importance; informally, it says that if all common symbols of two conservative modules are visible, then any model of both sets of visible theorems extends to a model of both sets of working theorems.

Theorem 16 *If modules $M = (\Phi, \Sigma, A)$ and $M' = (\Phi', \Sigma', A')$ are conservative and if $\Phi \cap \Phi' = \Sigma \cap \Sigma'$, then for any $(\Phi \cup \Phi')$-model m such that $m \models_{\Phi\cup\Phi'} \mathit{Vth}(M) \cup \mathit{Vth}(M')$ there is a $(\Sigma \cup \Sigma')$-model m' such that $m'\restriction_{\Phi\cup\Phi'} = m$ and $m' \models_{\Sigma\cup\Sigma'} \mathit{Th}(M) \cup \mathit{Th}(M')$.*

Proof: By the Satisfaction Condition, $m\restriction_\Phi \models_\Phi \mathit{Th}^\Sigma_\Phi(A)$ and $m\restriction_{\Phi'} \models_{\Phi'} \mathit{Th}^{\Sigma'}_{\Phi'}(A')$. Since (Φ, Σ, A) and (Φ', Σ', A') are conservative, there exist a Σ-model m_Σ of A and a Σ'-model $m_{\Sigma'}$ of A' such that $m_\Sigma\restriction_\Phi = m\restriction_\Phi$ and $m_{\Sigma'}\restriction_{\Phi'} = m\restriction_{\Phi'}$.


$$\begin{tikzcd}
 & & & \Phi \arrow[r, hook] \arrow[rd, hook] & \Sigma \arrow[rd, hook] & \\
\Sigma \cap \Sigma' & = & \Phi \cap \Phi' \arrow[ru, hook] \arrow[rd, hook] & & \Phi \cup \Phi' \arrow[r, hook] & \Sigma \cup \Sigma' \\
 & & & \Phi' \arrow[r, hook] \arrow[ru, hook] & \Sigma' \arrow[ru, hook] &
\end{tikzcd}$$

Then by the functoriality of reducts, $m_\Sigma\restriction_{\Phi\cap\Phi'} = (m_\Sigma\restriction_\Phi)\restriction_{\Phi\cap\Phi'} = (m\restriction_\Phi)\restriction_{\Phi\cap\Phi'} = m\restriction_{\Phi\cap\Phi'} = (m\restriction_{\Phi'})\restriction_{\Phi\cap\Phi'} = (m_{\Sigma'}\restriction_{\Phi'})\restriction_{\Phi\cap\Phi'} = m_{\Sigma'}\restriction_{\Phi\cap\Phi'}$. Since $\Sigma \cap \Sigma' = \Phi \cap \Phi'$, and **Mod** preserves intersection-union pushouts, and by the construction of pullbacks in **Cat**, there is a (unique) $(\Sigma \cup \Sigma')$-model m' such that $m' \lceil_\Sigma = m_\Sigma$ and $m' \lceil_{\Sigma'} = m_{\Sigma'}$; thus $m' \lceil_\Sigma \models_\Sigma A$ and $m' \lceil_{\Sigma'} \models_{\Sigma'} A'$. Therefore the Satisfaction Condition gives $m' \models_{\Sigma \cup \Sigma'} A \cup A'$. The reader may check that this essentially says that m' is a model of both $Th(M)$ and $Th(M')$, and that $(m' \lceil_{\Phi \cup \Phi'}) \lceil_\Phi = m \lceil_\Phi$ and $(m' \lceil_{\Phi \cup \Phi'}) \lceil_{\Phi'} = m \lceil_{\Phi'}$. Therefore $m' \lceil_{\Phi \cup \Phi'}$ satisfies the conditions that are uniquely satisfied by m (because the union of Φ and Φ' is a pushout of their intersection, because **Mod** preserves it, and because of the way pullbacks are built in **Cat**). Therefore $m' \lceil_{\Phi \cup \Phi'} = m$. □

Proposition 17 *If M and M' are modules as in Theorem 16, then*

1. $[\![M + M']\!]_1 = [\![M + M']\!]_2$.
2. $m \models M + M'$ *iff* $m \lceil_\Phi \models M$ *and* $m \lceil_{\Phi'} \models M'$ *for any* $(\Phi \cup \Phi')$*-model* m.
3. $M + M'$ *is conservative.*

Proof: 1. is equivalent to $Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup A') = Th^{\Phi \cup \Phi'}_{\Phi \cup \Phi'}(Th^\Sigma_\Phi(A) \cup Th^{\Sigma'}_{\Phi'}(A'))$. By Proposition 8, $Th^\Sigma_\Phi(A) \subseteq Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup B)$ and $Th^{\Sigma'}_{\Phi'}(B) \subseteq Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup B)$, so $Th^{\Phi \cup \Phi'}_{\Phi \cup \Phi'}(Th^\Sigma_\Phi(A) \cup Th^{\Sigma'}_{\Phi'}(B)) \subseteq Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup B)$. Conversely, consider a $(\Phi \cup \Phi')$-sentence a such that $A \cup A' \models_{\Sigma \cup \Sigma'} a$, and let m be a $(\Phi \cup \Phi')$-model for $Th^\Sigma_\Phi(A)$ and $Th^{\Sigma'}_{\Phi'}(A')$. By Theorem 16, there exists a $(\Sigma \cup \Sigma')$-model m' of $A \cup A'$ such that $m' \lceil_{\Phi \cup \Phi'} = m$. Then $m' \models_{\Sigma \cup \Sigma'} a$, and so by the Satisfaction Condition, $m' \lceil_{\Phi \cup \Phi'} \models_{\Phi \cup \Phi'} a$, that is, $m \models_{\Phi \cup \Phi'} a$. Consequently, a is in $Th^{\Phi \cup \Phi'}_{\Phi \cup \Phi'}(Th^\Sigma_\Phi(A) \cup Th^{\Sigma'}_{\Phi'}(A'))$. This shows that $Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup A') \subseteq Th^{\Phi \cup \Phi'}_{\Phi \cup \Phi'}(Th^\Sigma_\Phi(A) \cup Th^{\Sigma'}_{\Phi'}(A'))$.

2. follows from the equivalences:

$$\begin{array}{lll}
m \models M + M' & \text{iff} & \\
m \models_{\Phi \cup \Phi'} Vth(M + M') & \text{iff} & \text{(by 1.)} \\
m \models_{\Phi \cup \Phi'} (Vth(M) \cup Vth(M'))^\bullet & \text{iff} & \\
m \models_{\Phi \cup \Phi'} Vth(M) \cup Vth(M') & \text{iff} & \\
m \models_{\Phi \cup \Phi'} Vth(M) \text{ and } m \models_{\Phi \cup \Phi'} Vth(M') & \text{iff} & \\
m \lceil_\Phi \models_\Phi Vth(M) \text{ and } m \lceil_{\Phi'} \models_{\Phi'} Vth(M') & \text{iff} & \\
m \lceil_\Phi \models M \text{ and } m \lceil_{\Phi'} \models Vth(M')\ . & &
\end{array}$$

Only the right to left implication is interesting, since the other direction needs neither conservativity nor that $\Phi \cap \Phi' = \Sigma \cap \Sigma'$.

For 3., take a $(\Phi \cup \Phi')$-model m of $Th^{\Sigma \cup \Sigma'}_{\Phi \cup \Phi'}(A \cup A')$. Then m is also a $(\Phi \cup \Phi')$-model of $Th^\Sigma_\Phi(A)$ and $Th^{\Sigma'}_{\Phi'}(A')$, and by Theorem 16 there is a $(\Sigma \cup \Sigma')$-model m' of $A \cup A'$ such that $m' \lceil_{\Phi \cup \Phi'} = m$. Therefore, m' is a $(\Sigma \cup \Sigma')$-model of $Th^{\Sigma \cup \Sigma'}_{\Sigma \cup \Sigma'}(A \cup A')$; this shows that $(\Phi, \Sigma, A) + (\Phi', \Sigma', A')$ is conservative. □

Despite its somewhat complex proof, this result looks so natural that one might be tempted to think its hypothesis too strong. But visibility of shared symbols really is needed. Let M have a visible constant 0, a private constant c and a sentence $0 = c$, while M' has a constant 1, the same private c, and the sentence $1 = c$. Then $(\mathit{Vth}(M) \cup \mathit{Vth}(M'))^\bullet$ is empty while $\mathit{Vth}(M{+}M')$ contains $0 = 1$. This suggests that an implementation of aggregation should rename all shared private symbols even if they occurred as a consequence of enriching a shared module. Conservativity of M and M' is also needed. Let Φ be the signature with constants $0, 1$ and a binary operation $+$, let Σ add to Φ a private constant c, let A have the equation $0 + c = 1 + c$, let $\Sigma' = \Phi' = \Phi$, and let A' have equations $(\forall X)\ X + X = 0$ and $(\forall X, Y, Z)\ X + (Y + Z) = (X + Y) + Z$. Then the equation $0 = 1$ is in $\mathit{Vth}(M + M')$ but not in $(\mathit{Vth}(M) \cup \mathit{Vth}(M'))^\bullet$, and this occurs because M is not conservative.

The two semantics naturally extend to an arbitrary number of modules: $[\![M_1 + \cdots + M_n]\!]_1 = \mathit{Vth}(M_1 + \cdots + M_n)$, and $[\![M_1 + \cdots + M_n]\!]_2 = (\mathit{Vth}(M_1) \cup \ldots \cup \mathit{Vth}(M_n))^\bullet$, where the closure is over $\Phi_1 \cup \ldots \cup \Phi_n$. We then have

Corollary 18 *If the given institution is distributive and if $M_j = (\Phi_j, \Sigma_j, A_j)$ for $j = 1, ..., n$ are conservative such that $\Sigma_i \cap \Sigma_j = \Phi_i \cap \Phi_j$ for $i, j = 1, ..., n$ with $M_i \neq M_j$, then:*

1. $[\![M_1 + \cdots + M_n]\!]_1 = [\![M_1 + \cdots + M_n]\!]_2$.
2. *$m \models M_1 + \cdots + M_n$ iff $m{\restriction}_{\Phi_j} \models M_j$ for $j = 1, ..., n$, where m is a $(\Phi_1 \cup \ldots \cup \Phi_n)$-model.*
3. *$M_1 + \cdots + M_n$ is conservative.*

5.2 Renaming

Renaming is straightforward for transparent algebraic specifications: given a specification (Σ, A) and a morphism $h\colon \Sigma \to \Sigma'$, the *renaming of* (Σ, A) *by* h, denoted $(\Sigma, A)\star h$, is obtained by renaming each Σ-sentence in A, i.e., $(\Sigma, A)\star h = (\Sigma', h(A))$. The situation is more complex for modules with hiding, because only the visible symbols are renamed, and because renamed symbols could clash with private names. These problems are handled abstractly by signature pushouts.

Definition 19 Given $M = (\Phi, \Sigma, A)$ and $h\colon \Phi \to \Phi'$, **the renaming of** M **by** h, written $M \star h$, is the module $(\Phi', \Sigma_h, h_\Sigma(A))$ (see Convention 5). Moreover, $[\![M \star h]\!]_1 = \mathit{Vth}(\Phi', \Sigma_h, h_\Sigma(A))$, and $[\![M \star h]\!]_2 = h(\mathit{Vth}(M))^\bullet$, where the closure is over Φ'. ■

The morphism h is first extended to the morphism h_Σ on the whole working signature, and then A is renamed by h_Σ. This is well defined because $\Phi' \hookrightarrow \Sigma_h$ and $h_\Sigma(A)$ is a set of Σ_h-sentences. Notice that $h\colon M \to M \star h$ is a module morphism by the Generalized Closure Lemma (Lemma 9), so $m \models M \star h$ implies $m{\restriction}_h \models M$ for any Φ_h-model m.

The next result says that, assuming conservativity, the two semantics coincide: the visible theorems of a renamed module are exactly those generated by the renamed visible theorems of the initial module, i.e., the models of the renamed module are *exactly* those whose reducts are models of the initial module; moreover, conservativity is preserved under renaming.

Proposition 20 *If $M = (\Phi, \Sigma, A)$ is a conservative module, then*

1. $[\![M \star h]\!]_1 = [\![M \star h]\!]_2$ *.*

2. $m \models M \star h$ *iff* $m{\restriction}_h \models M$ *for any* Φ'*-model* m*.*

3. $M \star h$ *is conservative.*

Proof: For 1., we need $Th^{\Phi'}_{\Phi'}(h(Th^{\Sigma}_{\Phi}(A))) = Th^{\Sigma_h}_{\Phi'}(h_\Sigma(A))$. By Lemma 9, $h(Th^{\Sigma}_{\Phi}(A))) \subseteq Th^{\Sigma_h}_{\Phi'}(h_\Sigma(A))$; applying $Th^{\Phi'}_{\Phi'}$ to this inclusion, Proposition 8 gives $Th^{\Phi'}_{\Phi'}(h(Th^{\Sigma}_{\Phi}(A))) \subseteq Th^{\Sigma_h}_{\Phi'}(h_\Sigma(A))$. Conversely, let $a \in \mathbf{Sen}(\Phi')$ such that $h_\Sigma(A) \models_{\Sigma_h} a$ and let m' be a Φ'-model such that $m' \models_{\Phi'} h(Th^{\Sigma}_{\Phi}(A))$. We need to show $m' \models_{\Phi'} a$. By the Satisfaction Condition, $m'{\restriction}_h \models_{\Phi} Th^{\Sigma}_{\Phi}(A)$. Since (Φ, Σ, A) is conservative, there is a Σ-model m such that $m{\restriction}_{\Phi} = m'{\restriction}_h$ and $m \models_\Sigma Th^{\Sigma}_{\Sigma}(A)$. By the construction of pullbacks in **Cat** (Section 2.1), and since **Mod** preserves intersection-union pushouts (Definition 3), there is a Σ_h-model, say m_h, such that $m_h{\restriction}_{h_\Sigma} = m$ and $m_h{\restriction}_{\Phi'} = m'$. Then $m_h{\restriction}_{h_\Sigma} \models_\Sigma A$ (because $m \models_\Sigma Th^{\Sigma}_{\Sigma}(A)$ and $A \subseteq Th^{\Sigma}_{\Sigma}(A)$), and so $m_h \models_{\Sigma_h} h_\Sigma(A)$. Further, $m_h \models_{\Sigma_h} a$ because $h_\Sigma(A) \models_{\Sigma_h} a$. Finally, if $\imath'$ is the inclusion $\Phi' \hookrightarrow \Sigma_h$ then $m_h \models_{\Sigma_h} \imath'(a)$, so $m_h{\restriction}_{\imath'} \models_{\Phi'} a$; therefore $m' \models_{\Phi'} a$.

2. follows since $m{\restriction}_h \models M$ iff $m{\restriction}_h \models_\Phi Vth(M)$ iff $m \models_{\Phi'} h(Vth(M))$ (Satisfaction Condition) iff $m \models_{\Phi'} h(Vth(M))^\bullet$ iff $m \models_{\Phi'} Vth(M \star h)$ (by 1.) iff $m \models M \star h$.

For 3., let m be a Φ'-model of $Th^{\Sigma_h}_{\Phi'}(h_\Sigma(A))$, that is, $m \models M \star h$. By 2., $m{\restriction}_h$ is a Φ-model of $Th^{\Sigma}_{\Phi}(A)$. Then by conservativity of (Φ, Σ, A), there is a Σ-model m_Σ of A such that $m_\Sigma{\restriction}_\Phi = m{\restriction}_h$. But the pair of morphisms h_Σ and $\Phi' \hookrightarrow \Sigma_h$ is a pushout of h and $\Phi \hookrightarrow \Sigma$; therefore, since **Mod** preserves these pushouts (see Definition 3), by the construction of pullbacks in **Cat**, there is a Σ_h-model m' such that $m'{\restriction}_{(h_\Sigma)} = m_\Sigma$ and $m'{\restriction}_{\Phi'} = m$. Then by the Satisfaction Condition, $m' \models_{\Sigma_h} h_\Sigma(A)$, that is m' is a Σ_h-model of $Th^{\Sigma_h}_{\Sigma_h}(h_\Sigma(A))$. Therefore, for a Φ'-model m of $Vth(M \star h)$ we have a Σ_h-model m' of $Th(M \star h)$ such that $m'{\restriction}_{\Phi'} = m$. This shows that $M \star h$ is conservative. □

One might think that conservativity is not needed here. But consider the unsorted equational logic module $M = (\Phi, \Sigma, A)$ where Φ contains constants a, b and a binary operation f, Σ adds one more constant c, and A contains the equations $f(a, c) = a$ and $f(b, c) = f(a, a)$; suppose also that Φ' consists of only one constant d and that h takes both a and b to d. Then $h(Vth(M))^\bullet$ is an empty theory because $Vth(M)$ is empty, while $Vth(M \star h)$ contains the equation $f(d, d) = d$. Notice that M is not conservative.

A desirable property of renamings is that they can be composed, in the sense that $(M \star h) \star g = M \star (h; g)$ for any appropriate h and g. This is straightforward for transparent specifications, but it can be hard to insure when hiding is allowed because of the variety of conventions for renaming hidden symbols to prevent name clashes with the visible symbols in the result (this is similar to the variety of choices for h_Σ in Convention 5).

Proposition 21 *If pushouts of inclusions in* **Sign** *are chosen such that they* *can be composed vertically (see Section 3), then* $(M \star h) \star g = M \star (h; g)$ *for any module* $M = (\Phi, \Sigma, A)$ *and any morphisms* $h\colon \Phi \to \Phi'$ *and* $g\colon \Phi' \to \Phi''$.

Proof: $((\Phi, \Sigma, A) \star h) \star g$ equals $(\Phi', \Sigma_h, h_\Sigma(A)) \star g$ by Definition 19, which equals $(\Phi'', (\Sigma_h)_g, g_{(\Sigma_h)}(h_\Sigma(A)))$ also by Definition 19, which further equals $(\Phi'', \Sigma_{h;g}, (h; g)_\Sigma(A))$ by hypothesis, which finally equals $(\Phi, \Sigma, A) \star (h; g)$, again by Definition 19. □

5.3 Enrichment

A common way to reuse software and specification is through enrichment, which adds functionality to an existing module. For example, LILEANNA [26] implements enrichment by adding a partial signature to the given signature and then adding code over the resulting signature. However, it is simpler to use extensions of total signatures.

Definition 22 Given modules $M = (\Phi, \Sigma, A)$ and (Φ', Σ', A') with $\Phi \hookrightarrow \Phi'$ and $\Sigma \hookrightarrow \Sigma'$, the **enrichment of** M **by** (Φ', Σ', A'), written $M@(\Phi', \Sigma', A')$, is the module $(\Phi', \Sigma', A \cup A')$, and $[\![M@(\Phi', \Sigma', A')]\!]_1 = \mathit{Vth}(\Phi', \Sigma', A \cup A')$ and $[\![M@(\Phi', \Sigma', A')]\!]_2 = \mathit{Vth}(\Phi', \Phi' \cup \Sigma, A \cup \mathit{Vth}(\Phi' \cup \Sigma, \Sigma', A'))$. ■

Both visible (Φ') and private (Σ') symbols can be added, as well as sentences (A') involving all these symbols. Note that if $\imath$ is the inclusion $\Phi \hookrightarrow \Phi'$ then $\imath\colon M \to M@(\Phi', \Sigma', A')$ is a morphism of modules, so $m \models M@(\Phi', \Sigma', A')$ implies $m\restriction_\Phi \models M$ for any Φ'-model m.

The first semantics is straightforward, but the second requires some explanation. The key is to take a *working-in-M* perspective, similar to the intuition for module enrichment in software engineering, and to consider how the newly added features affect the semantics of the current working environment, regarded as visible. Since new visible symbols are added to M, those symbols extend the working signature to $\Phi' \cup \Sigma$, and their effect on the working environment is the visible theorems of the module $(\Phi' \cup \Sigma, \Sigma', A')$. We first prove the following:

Lemma 23 *In the context of Definition 22, if* $\Psi \hookrightarrow \Sigma'$ *is such that* (Ψ, Σ', A') *is conservative, then* $\mathit{Th}_\Psi^{\Sigma'}(A \cup A') = \mathit{Th}_\Psi^{\Psi}(A \cup \mathit{Th}_\Psi^{\Sigma'}(A'))$.

Proof: Since $A \subseteq \mathit{Th}_\Psi^{\Sigma}(A \cup A')$ and $\mathit{Th}_\Psi^{\Sigma}(A') \subseteq \mathit{Th}_\Psi^{\Sigma'}(A \cup A')$, it follows by Proposition 8 that $\mathit{Th}_\Psi^{\Psi}(A \cup \mathit{Th}_\Psi^{\Sigma'}(A')) \subseteq \mathit{Th}_\Psi^{\Sigma'}(A \cup A')$. Conversely, let a be a (Ψ)-sentence in $\mathit{Th}_\Psi^{\Sigma'}(A \cup A')$. In order to prove that a is in $\mathit{Th}_\Psi^{\Psi}(A \cup \mathit{Th}_\Psi^{\Sigma'}(A'))$, take a (Ψ)-model m of $A \cup \mathit{Th}_\Psi^{\Sigma'}(A')$. Since (Ψ, Σ', A') is conservative, there is a Σ'-model m' of A' such that $m'\restriction_\Psi = m$. But $m \models_\Psi A$, that is, $m'\restriction_\Psi \models_\Psi A$; then by the satisfaction condition we get $m' \models_{\Sigma'} A$. Therefore $m' \models_{\Sigma'} A \cup A'$, and so $m' \models_{\Sigma'} a$, because we supposed that $A \cup A' \models_{\Sigma'} a$. Consequently, the satisfaction condition implies $m'\restriction_\Psi \models_\Psi a$, i.e., $m \models_\Psi a$. Therefore, a is in $\mathit{Th}_\Psi^{\Psi}(A \cup \mathit{Th}_\Psi^{\Sigma'}(A'))$. □

Proposition 24 *In the context of Definition 22, if* $(\Phi' \cup \Sigma, \Sigma', A')$ *is conservative then*

1. $[\![M@(\Phi', \Sigma', A')]\!]_1 = [\![M@(\Phi', \Sigma', A')]\!]_2$,
2. $M@(\Phi', \Sigma', A')$ *is conservative if* $(\Phi', \Phi' \cup \Sigma, A \cup \mathit{Vth}(\Phi' \cup \Sigma, \Sigma', A'))$ *is conservative.*

Proof: Replacing Ψ by $\Phi' \cup \Sigma$ and then taking the Φ'-visible theorems of the two sides in the equality given by Lemma 23, we get $\mathit{Th}^{\Sigma'}_{\Phi'}(A \cup A') = \mathit{Th}^{\Phi' \cup \Sigma}_{\Phi'}(A \cup \mathit{Th}^{\Sigma'}_{\Phi' \cup \Sigma}(A'))$. Equation 1. now follows from the calculation

$$\begin{aligned}
[\![M@(\Phi', \Sigma', A')]\!]_1 &= \mathit{Vth}(\Phi', \Sigma', A \cup A') \\
&= \mathit{Th}^{\Sigma'}_{\Phi'}(A \cup A') \\
&= \mathit{Th}^{\Phi' \cup \Sigma}_{\Phi'}(A \cup \mathit{Th}^{\Sigma'}_{\Phi \cup \Sigma}(A')) \\
&= \mathit{Vth}(\Phi', \Phi' \cup \Sigma, A \cup \mathit{Vth}(\Phi' \cup \Sigma, \Sigma', A')) \\
&= [\![M@(\Phi', \Sigma', A')]\!]_2 \ .
\end{aligned}$$

For 2., let m be a Φ'-model of $\mathit{Vth}(M@(\Phi', \Sigma', A'))$. Then by Proposition 8, m is also a Φ'-model of $\mathit{Vth}(\Phi', \Phi' \cup \Sigma, A \cup \mathit{Vth}(\Phi' \cup \Sigma, \Sigma', A'))$ and so by conservativity, there is a $(\Phi' \cup \Sigma)$-model m'' of $A \cup \mathit{Vth}(\Phi' \cup \Sigma, \Sigma', A')$ such that $m''\lceil_{\Phi'} = m$. Now, since $(\Phi' \cup \Sigma, \Sigma', A')$ is conservative, there is a Σ'-model m' of A' such that $m'\lceil_{\Phi' \cup \Sigma} = m''$. By the Satisfaction Condition, $m' \models_{\Sigma'} A$, so that $m' \models_{\Sigma'} A \cup A'$ and, of course, $m'\lceil_{\Phi'} = m$. □

One can enrich an imported module with essentially anything, including inconsistent sentences. But an important special case is when no new visible symbols are added. This is useful when refining an incomplete module that declares an interface, or when one wants to further constrain an existing module in order to change its intended semantics (for example, adding the equation $10 = 0$ to the module that specifies integers to get integers modulo 10).

Corollary 25 *If* $M' = (\Sigma, \Sigma', A')$ *is a conservative module, then*

1. $\mathit{Vth}(M@(\Phi, \Sigma', A')) = \mathit{Vth}(\Phi, \Sigma, A \cup \mathit{Vth}(M'))$*, and*
2. $M@(\Phi, \Sigma', A')$ *is conservative if* $(\Phi, \Sigma, A \cup \mathit{Vth}(M'))$ *is conservative.*

Technically, enriching is a special case of aggregation in our approach, because $M@(\Phi', \Sigma', A') = M + (\Phi', \Sigma', A')$. However, the results that were developed for aggregation assumed that the aggregated modules did not have common private symbols, which fails for enrichment, where all the private symbols of the enriched module are available.

5.4 Hiding

Hiding information is very natural in our approach; it just restricts visibility to a deeper subsignature.

Definition 26 If $M = (\Phi, \Sigma, A)$ is a module and Ψ is a subsignature of Φ, then $\Psi \Box M$ is the module (Ψ, Σ, A); also $[\![\Psi \Box M]\!]_1 = \mathit{Vth}(\Psi, \Sigma, A)$, and $[\![\Psi \Box M]\!]_2 = (\Psi, \Phi, \mathit{Vth}(M))$. We call $\Box$ the **information hiding operator**. ■

Fewer theorems remain visible after an information hiding operation. The term "export operator" is used for $\Box$ in [1], but we prefer the more explicit term, after [9]. If $\imath\colon \Psi \hookrightarrow \Phi$, then $\imath\colon \Psi\Box M \to M$ is a module morphism, so $m \models M$ implies $m\restriction_\Psi \models \Psi\Box M$ for any Φ-model m. The following shows the relationship between the visible theorems of $\Psi\Box M$ and the visible theorems of M, that is, a relationship between the two semantics, and it also gives a sufficient condition under which hiding preserves conservativity.

Proposition 27 *If $M = (\Phi, \Sigma, A)$ is a module and $\Psi \hookrightarrow \Phi$ is a signature inclusion, then*

1. $[\![\Psi\Box M]\!]_1 = [\![\Psi\Box M]\!]_2$ *and*
2. $\Psi\Box M$ *is conservative if M and $(\Psi, \Phi, Vth(M))$ are conservative.*

Proof: 1. is equivalent to $Th^\Sigma_\Psi(A)) = Th^\Phi_\Psi(Th^\Sigma_\Phi(A))$, which is 10. of Proposition 8. For 2., let m be a Ψ-model of $Th^\Sigma_\Psi(A))$. Since $Th^\Sigma_\Psi(A)) = Th^\Phi_\Psi(Th^\Sigma_\Phi(A))$, by the conservativity of $(\Psi, \Phi, Th^\Sigma_\Phi(A))$, there is a Φ-model m' of $Th^\Phi_\Phi(Th^\Sigma_\Phi(A)) = Th^\Sigma_\Phi(A)$ such that $m'\restriction_\Psi = m$. Then by the conservativity of (Φ, Σ, A) there is a Σ-model m'' of $Th^\Sigma_\Sigma(A)$ with $m''\restriction_\Phi = m'$. Therefore $m''\restriction_\Psi = m$, and so (Φ, Σ, A) is conservative. $\Box$

Although conservativity was not needed to show equivalence of the two semantics for hiding, it is needed for equivalence of the semantics of the other operations. This is why we always give sufficient conditions for the conservativity of resulting modules. Notice that conservativity of M does not guarantee conservativity of $\Psi\Box M$: for example, one can take M to be a transparent module, which is automatically conservative, and Ψ such that $\Psi\Box M$ is not conservative (as in Example 13).

Testing conservativity of a module (Ψ, Σ, A) can be difficult, and depends on the underlying logic. In many sorted equational logics, one can enrich a Ψ-algebra with new carriers for private sorts (in $\Sigma - \Psi$), and with new private operations, and then show that the new Σ-algebra satisfies A. Of course, the fewer private symbols, the easier this is. For this reason, we prefer to reduce showing the conservativity of a module with visible signature Ψ and working signature Σ, to the conservativity of other two modules: one with visible signature Ψ and working signature Φ, for $\Psi \hookrightarrow \Phi \hookrightarrow \Sigma$, and the other with visible signature Φ and working signature Σ, as in the above proposition.

5.5 Parameterization

One of the most effective supports for software reuse is parameterization. Many expositions only treat the one parameter case, saying that it generalizes to many parameters in an obvious way. Since one of our goals is conditions for the correctness of logic-independent algorithms to flatten complex module structures, and since shared features of parameters are important in this, we treat multiple parameterization explicitly, and prove that it is a colimit.

Definition 28 A **parameterized module** $M[\alpha_1 :: P_1, ..., \alpha_n :: P_n]$ is a set of module morphisms $\alpha_j ; \imath_j : P_j \to M$, where $M = (\Phi, \Sigma, A)$ and $P_j = (\Phi'_j, \Sigma'_j, B_j)$ for $j = 1, ..., n$, such that:

- $\alpha_j : \Phi'_j \to \Phi_j$ are isomorphisms of signatures,
- $\imath_j : \Phi_j \hookrightarrow \Phi$ are inclusions of signatures, and
- $\Phi_1, ..., \Phi_n$ are disjoint.

We say that M is **parameterized** by $\alpha_1, ..., \alpha_n$. $P_1, ..., P_n$ are called the **interfaces** and M the **body**. Given a parameterized module $M[\alpha_1 :: P_1, ..., \alpha_n :: P_n]$ and morphisms $h_j : P_j \to M_j$ with $M_j = (\Psi_j, \Omega_j, A_j)$ for $j = 1, ..., n$, the **instantiation of** M **by** $h_1, ..., h_n$, written $M[h_1, ..., h_n]$, is the module

$$(\Phi_h,\ \Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^{n} \Omega_j,\ (h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^{n} A_j)\ ,$$

where $h = \bigcup_{j=1}^{n} \alpha_j^{-1} ; h_j$ (see Section 2.2 and Convention 5). ■

$$\begin{tikzcd}
\Phi_i \arrow[r, hook] \arrow[d, "\alpha_i^{-1};h_i"'] & \bigcup_{j=1}^{n} \Phi_j \arrow[r, hook] \arrow[d, "h"] & \Phi \arrow[r, hook] \arrow[d, "h_\Phi"'] & \Sigma \arrow[d, "(h_\Phi)_\Sigma"] \\
\Psi_i \arrow[r, hook] & \bigcup_{j=1}^{n} \Psi_j \arrow[r, hook] & \Phi_h \arrow[r, hook] & \Sigma_{(h_\Phi)}
\end{tikzcd}$$

This apparently abstract and complex definition has a natural interpretation. First, it says that parameters are regarded as having disjoint interface signatures in the parameterized module, which is natural to avoid obvious name clashes (in practice, parameter signatures can be automatically made disjoint by tagging them with their parameter names). Second, it informally says that the instantiation of a module is computed as follows:

1. Calculate $\alpha_i^{-1}; h_i$, which gives for each symbol in M belonging to a parameter P_i its actual instance symbol;
2. store all these mappings in a table h;
3. Calculate h's pushout h_Φ, which "knows" how to avoid name clashes between visible symbols defined in M and visible symbols that may accidentally occur in some of the actual parameters;
4. Calculate h_Φ's pushout $(h_\Phi)_\Sigma$, which solves further name conflicts with M's private symbols; and
5. Rename all sentences declared in M accordingly.

Notice that all these steps are purely textual and can be efficiently implemented.

Proposition 29 *In the context of Definition 28,*

1. $h_\Phi : M \to M[h_1, ..., h_n]$ *is a module morphism,*
2. $\bigcup_{j=1}^{n} \Psi_j \hookrightarrow \Phi_h : M_1 + \cdots + M_n \to M[h_1, ..., h_n]$ *is also a morphism, and*
3. $M[h_1, ..., h_n] = M \star h_\Phi + M_1 + \cdots + M_n$.

Proof: For 1., $h_\varphi : M \to M[h_1, ..., h_n]$ is a morphism because:

$$\begin{array}{rcll} h_\Phi(\mathit{Vth}(M)) & = & h_\Phi(Th^{\Sigma}_{\Phi}(A)) & \\ & \subseteq & Th^{\Sigma_{(h_\Phi)}}_{\Phi_h}((h_\Phi)_\Sigma(A)) & \text{(Lemma 9)} \\ & \subseteq & Th^{\Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^n \Omega_j}_{\Phi_h}((h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \text{(Proposition 8)} \\ & = & \mathit{Vth}(M[h_1, ..., h_n]) \ . & \end{array}$$

2. is straightforward, because by Proposition 8,

$$Th^{\bigcup_{j=1}^n \Omega_j}_{\bigcup_{j=1}^n \Psi_j}(\bigcup_{j=1}^n A_j) \ \subseteq \ Th^{\Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^n \Omega_j}_{\Phi_h}((h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) \ .$$

3. follows from the equalities

$$\begin{array}{cll} & M \star h_\Phi + M_1 + \cdots + M_n & \\ = & (\Phi, \Sigma, A) \star h_\Phi + (\Psi_1, \Omega_1, A_1) + \cdots + (\Psi_n, \Omega_n, A_n) & \\ = & (\Phi_h, \Sigma_{(h_\Phi)}, (h_\Phi)_\Sigma(A)) + (\Psi_1, \Omega_1, A_1) + \cdots + (\Psi_n, \Omega_n, A_n) & \text{(Def. 19)} \\ = & (\Phi_h \cup \bigcup_{j=1}^n \Psi_j, \Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^n \Omega_j, (h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \text{(Def. 14)} \\ = & (\Phi_h, \Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^n \Omega_j, (h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \\ = & M[h_1, ..., h_n] \ , & \text{(Def. 28)} \end{array}$$

using Proposition 8. □

This proposition suggests the following for the two semantics of instantiating a parameterized module:

Definition 30 Using the same context and notation as in Definition 28, let $[\![M[h_1, ..., h_n]]\!]_1 = \mathit{Vth}(\Phi_h, \Sigma_{(h_\Phi)} \cup \bigcup_{j=1}^n \Omega_j, (h_\Phi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j)$ and also let $[\![M[h_1, ..., h_n]]\!]_2 = (h_\Phi(\mathit{Vth}(M)) \cup \bigcup_{j=1}^n \mathit{Vth}(M_j))^\bullet$, where closure is over Φ_h. ■

The following gives sufficient and necessary conditions under which the two semantics of the instantiated module coincide, and shows that conservativity of the instantiated module does not depend on conservativity of its original interface:

Proposition 31 *In the context of Definition 28, if the given institution is distributive and if*

- $M, M_1, ..., M_n$ *are conservative,*
- $\Sigma_{(h_\Phi)} \cap \Omega_j = \Psi_j$ *for* $j = 1, ..., n$, *and*
- $\Omega_i \cap \Omega_j = \Psi_i \cap \Psi_j$ *for* $i, j = 1, ..., n$, *with* $M_i \neq M_j$,

then

1. $[\![M[h_1, ..., M_n]]\!]_1 = [\![M[h_1, ..., M_n]]\!]_2$,
2. $m \models M[h_1, ..., h_n]$ *iff* $m\restriction_{h_\Phi} \models M$ *and* $m\restriction_{\Phi_j} \models M_j$ *for any* Φ_h*-model* m,
3. $M[h_1, ..., h_n]$ *is conservative.*

Proof: By Proposition 20, $Vth(M \star h_\Phi) = h_\Phi(Vth(M))^\bullet$ and $M \star h_\Phi$ is conservative, where the closure is over Φ_h-sentences. Since $M[h_1, ..., h_n] = M \star h_\Phi + M_1 + \cdots + M_n$, iteratively applying Proposition 17 we get that $Vth(M[h_1, ..., h_n]) = (h_\Phi(Vth(M))^\bullet \cup \bigcup_{j=1}^n Vth(M_j))^\bullet$ and $M[h_1, ..., h_n]$ is conservative, where the closures are over Φ_h-sentences. $\square$

The conservativity of $M, M_1, ..., M_n$ and all the equalities $\Omega_i \cap \Omega_j = \Psi_i \cap \Psi_j$ are needed because of 3. in Proposition 29 and because of their necessity in Propositions 17 and 20. The condition $\Sigma_{(h_\Phi)} \cap \Omega_j = \Psi_j$ may look restrictive, but a correct implementation of instantiation must consider it; informally, it says that either some private symbols in M should be renamed in the instantiated module to avoid conflicts with private symbols in M_j, or else that some symbols in M_j should be renamed before the actual instantiation takes place.

An important general property of parameterization is that the instantiated module is a colimit. This can be proved in a logic independent framework for modules which respect the above natural requirements:

Theorem 32 *In the context of Definition 28, if S_{ij} are $(\Psi_i \cap \Psi_j)$-modules*[3] *such that $Vth(S_{ij}) \subseteq Vth(M_i) \cap Vth(M_j)$, then $M[h_1, ..., h_n]$ is a colimit of*

$$\begin{tikzcd}
 & M_1 \arrow[d, dash, dotted] & P_1 \arrow[l, "h_1"'] \arrow[d, dash, dotted] \arrow[ddr, "\alpha_1;\imath_1"] & \\
 & M_i \arrow[dd, dash, dotted] & P_i \arrow[l, "h_i"'] \arrow[dd, dash, dotted] \arrow[dr, "\alpha_i;\imath_i"] & \\
S_{ij} \arrow[ur, hook, "\imath^i_{ij}"] \arrow[dr, hook, "\imath^j_{ij}"'] & & & M \\
 & M_j \arrow[d, dash, dotted] & P_j \arrow[l, "h_j"'] \arrow[d, dash, dotted] \arrow[ur, "\alpha_j;\imath_j"'] & \\
 & M_n & P_n \arrow[l, "h_n"'] \arrow[uur, "\alpha_n;\imath_n"'] &
\end{tikzcd}$$

where $\imath^i_{ij}$ is the inclusion $\Psi_i \cap \Psi_j \hookrightarrow \Psi_i$, for $i, j = 1, ..., n$.

Proof: Notice that $\imath^i_{ij} \colon S_{ij} \to M_i$ are module morphisms, and that giving a cocone of the diagram above is equivalent to giving a module C, a morphism $f \colon M \to C$, and morphisms $g_j \colon M_j \to C$ such that

- $h_i; g_i = \alpha_i; \imath_i; f$ for $i = 1, ..., n$, and
- $\imath^i_{ij}; g_i = \imath^j_{ij}; g_j$ for $i, j = 1, ..., n$.

[3] Think of S_{ij} as the shared modules of M_i and M_j.

The diagram below may help the reader follow the rest of this proof.

$$\begin{tikzcd}
\Phi_i \arrow[r, hook] \arrow[d, "\alpha_i^{-1};h_i"'] & \bigcup_{j=1}^n \Phi_j \arrow[r, hook] \arrow[d, "h"'] & \Phi \arrow[d, "h_\Phi"'] \arrow[ddr, "f"] & \\
\Psi_i \arrow[r, hook] \arrow[drrr, "g_i"'] & \bigcup_{j=1}^n \Psi_j \arrow[r, hook] \arrow[drr, dotted, "g"] & \Phi_h \arrow[dr, dotted, "r"] & \\
 & & & \Psi
\end{tikzcd}$$

First, we show that $h_\Phi\colon\ M \to M[h_1, ..., h_n]$ and $\Psi_i \hookrightarrow \Phi_h\colon\ M_i \to M[h_1, ..., h_n]$ is a cocone for $i = 1, ..., n$:

$$\begin{aligned}
& h_i; (\Psi_i \hookrightarrow \Phi_h) \\
= \;& (\alpha_i; \alpha_i^{-1}); h_i; ((\Psi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Psi_j); (\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h)) \\
= \;& \alpha_i; ((\alpha_i^{-1}; h_i); (\Psi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Psi_j)); (\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h) \\
= \;& \alpha_i; ((\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); h); (\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h) \\
= \;& \alpha_i; (\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); (h; (\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h)) \\
= \;& \alpha_i; (\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); ((\bigcup_{j=1}^n \Phi_j \hookrightarrow \Phi); h_\Phi) \\
= \;& \alpha_i; ((\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); (\bigcup_{j=1}^n \Phi_j \hookrightarrow \Phi)); h_\Phi \\
= \;& \alpha_i; \imath_i; h_\Phi \ .
\end{aligned}$$

Also, it is straightforward that $\imath_{ij}^i; (\Psi_i \hookrightarrow \Phi_h) = \imath_{ij}^j; (\Psi_j \hookrightarrow \Phi_h)$, because there is only one inclusion $\Psi_i \cap \Psi_j \hookrightarrow \Phi_h$.

Now let $f\colon\ M \to C$ and $g_i\colon\ M_i \to C$ for $i = 1, ..., n$ be another cocone, with $C = (\Psi, \Omega, B)$. Then Ψ together with the signature morphisms $g_i\colon\ \Psi_i \to \Psi$ for $i = 1, ..., n$ form a cocone in **Sign** for the diagram given by the pairs of inclusions

$$\begin{tikzcd}
\Psi_i & \Psi_i \cap \Psi_j \arrow[l, hook'] \arrow[r, hook] & \Psi_j
\end{tikzcd}$$

for $i, j = 1, ..., n$, so by 2. of Proposition 1, there is a unique signature morphism, let us call it $g\colon\ \bigcup_{j=1}^n \Psi_j \to \Psi$, such that $(\Psi_i \hookrightarrow \bigcup_{j=1}^n \Psi_j); g = g_i$. Since

$$\begin{aligned}
& (\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); ((\bigcup_{j=1}^n \Phi_j \hookrightarrow \Phi); f) \\
= \;& ((\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); (\bigcup_{j=1}^n \Phi_j \hookrightarrow \Phi)); f \\
= \;& \imath_i; f \\
= \;& (\alpha_i^{-1}; \alpha_i); \imath_i; f \\
= \;& \alpha_i^{-1}; (\alpha_i; \imath_i; f) \\
= \;& \alpha_i^{-1}; (h_i; g_i) \\
= \;& (\alpha_i^{-1}; h_i); g_i \\
= \;& (\alpha_i^{-1};; h_i); ((\Psi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Psi_j); g) \\
= \;& ((\alpha_i^{-1};; h_i); (\Psi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Psi_j)); g \\
= \;& ((\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); h); g \\
= \;& (\Phi_i \hookrightarrow \textstyle\bigcup_{j=1}^n \Phi_j); (h; g) \ ,
\end{aligned}$$

by 1. of Proposition 1, we get $(\bigcup_{j=1}^n \Phi_j \hookrightarrow \Phi); f = h; g$. But the rightmost square in the diagram at the beginning of this proof is a pushout, so there is a unique $r\colon \Phi_h \to \Psi$ such that $h_\Phi; r = f$ and $(\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h); r = g$.

We claim that r is a module morphism, from $M[h_1, ..., h_n]$ to C. Indeed,

$$\begin{array}{rcl} r(Vth(M[h_1, ..., h_n])) & = & r((h_\Phi(Vth(M)) \cup \bigcup_{j=1}^n Vth(M_j))^\bullet) \\ & \subseteq & r(h_\Phi(Vth(M)) \cup \bigcup_{j=1}^n Vth(M_j))^\bullet \\ & = & (r(h_\Phi(Vth(M))) \cup \bigcup_{j=1}^n r(Vth(M_j)))^\bullet \\ & = & (f(Vth(M)) \cup \bigcup_{j=1}^n g_j(Vth(M_j)))^\bullet \\ & \subseteq & Vth(C)^\bullet \\ & = & Vth(C)\ . \end{array}$$

The first assertion above follows by 1. of Proposition 31, and the second by the Closure Lemma. The uniqueness of $r\colon M[h_1, ..., h_n] \to C$ follows from the uniqueness of $r\colon \Phi_h \to \Phi$ as a signature morphism. Let $r'\colon M[h_1, ..., h_n] \to C$ be another morphism such that $h_\Phi; r' = f$ and $(\Psi_i \hookrightarrow \Phi_h); r' = g_i$ for $i = 1, ..., n$. Since the inclusions $\Psi_i \hookrightarrow \bigcup_{j=1}^n \Psi_j$ are an epimorphic family and

$$\begin{array}{rl} & (\Psi_i \hookrightarrow \bigcup_{j=1}^n \Psi_j); ((\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h); r') \\ = & ((\Psi_i \hookrightarrow \bigcup_{j=1}^n \Psi_j); (\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h)); r' \\ = & (\Psi_i \hookrightarrow \Phi_h); r' \\ = & g_i \\ = & (\Psi_i \hookrightarrow \bigcup_{j=1}^n \Psi_j); g\ , \end{array}$$

by 1. of Proposition 1, $(\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h); r' = g$. By the uniqueness of $r\colon \Phi_h \to \Psi$ with $h_\Phi; r = f$ and $(\bigcup_{j=1}^n \Psi_j \hookrightarrow \Phi_h); r = g$, it follows that $r' = r$. □

Many practical modules have just one parameter, so that sharing between actual parameters is not a problem, and a simpler result can be stated:

Corollary 33 *If $M[\alpha_1 :: P_1]$ is a parameterized module and if $h_1\colon P_1 \to M_1$ is a module morphism, then the square*

$$\begin{tikzcd} P_1 \arrow[r, "\alpha_1;\imath_1"] \arrow[d, "h_1"'] & M \arrow[d, "h_\varphi"] \\ M_1 \arrow[r, "\imath"'] & M[h_1] \end{tikzcd}$$

is a pushout in **MSpec**, *with $h = \alpha^{-1}; h_1$ and $\imath = \psi_1 \hookrightarrow \varphi_h$.*

Proof: By Theorem 32 with $S_{11} = M_1$. □

6 Conclusions and Future Research

This paper studies the composition of modules that can hide information, over inclusive institutions, which seem to include all logical systems of practical interest. Two different semantics for composed modules were defined, and it was shown they agree if all the involved modules are conservative. In addition, a number of basic "laws of software composition" were proved; these assert that two different module compositions have the same semantics, for all instances of their variables that range over modules. An important conclusion is that inclusive institutions can greatly simplify the kind of proofs that are done in this paper. This setting also allows algorithms for flattening compositions to be presented as mainly based on signatures pushouts, which is a purely textual operation for concrete institutions.

Interesting directions for future research include: extending the results to a multi-institutional framework (e.g., see [8, 25]), to accommodate multi-paradigm specification languages; proving further laws, such as distributivity (see [1, 9]); adapting the normalization theorem of [1] to our setting; and attempting to implement (e.g., in Perl) the composition operations in a way that would work for a wide range of specification and programming paradigms.

Acknowledgements. The authors thank Virgil Emil Căzănescu for his ongoing collaboration on inclusion systems, and Răzvan Diaconescu for stimulating debates on institution-based modularization.

Dedication. This paper is dedicated to Johan-Ole Dahl, a gentleman of the old school, and a pioneer in language design, whose work always exhibited the good taste and elegance of its author.

References

[1] Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

[2] Jan Bergstra and John Tucker. Equational specifications, complete term rewriting systems, and computable and semicomputable algebras. *Journal of the Association for Computing Machinery*, 42(6):1194–1230, 1995.

[3] Rod Burstall and Joseph Goguen. Putting theories together to make specifications. In Raj Reddy, editor, *Proceedings, Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058. Department of Computer Science, Carnegie-Mellon University, 1977.

[4] CoFI. CASL summary, 2000. `www.brics.dk/Projects/CoFi`.

[5] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 7(2):195–206, 1997.

[6] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems; part 2. *Journal of Universal Computer Science*, 6(1):5–21, 2000.

[7] Răzvan Diaconescu. Category-based modularization for equational logic programming. *Acta Informatica*, 33(5):477–510, 1996.

[8] Răzvan Diaconescu. Extra theory morphisms in institutions: logical semantics for multi-paradigm languages. *Applied Categorical Structures*, 6(4):427–453, 1998.

[9] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularization. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993.

[10] Joseph Goguen. Parameterized programming. *Transactions on Software Engineering*, SE–10(5):528–543, September 1984.

[11] Joseph Goguen. Principles of parameterized programming. In Ted Biggerstaff and Alan Perlis, editors, *Software Reusability, Volume I: Concepts and Models*, pages 159–225. Addison Wesley, 1989.

[12] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992.

[13] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000.

[14] Joseph Goguen and Grigore Roşu. Hiding more of hidden algebra. In *Formal Methods 1999 (FM'99)*, volume 1709 of *Lecture Notes in Computer Sciences*, pages 1704–1719. Springer-Verlag, 1999.

[15] Joseph Goguen and Grigore Roşu. Institution morphisms. *Formal Aspects of Computing*, 2002. To appear.

[16] Joseph Goguen and William Tracz. An implementation-oriented semantics for module composition. In Gary Leavens and Murali Sitaraman, editors, *Foundations of Component-based Systems*, pages 231–263. Cambridge, 2000.

[17] Horst Herrlich and George Strecker. *Category Theory.* Allyn and Bacon, 1973.

[18] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

[19] José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

[20] Till Mossakowski. Specifications in an arbitrary institution with symbols. In *Proceedings, WADT'99*, volume 1827 of *Lecture Notes in Computer Science*, pages 252–270. Springer, 2000.

[21] David Parnas. Information distribution aspects of design methodology. *Information Processing '72*, 71:339–344, 1972. Proceedings of 1972 IFIP Congress.

[22] Grigore Roşu. *Hidden Logic.* PhD thesis, University of California at San Diego, 2000. `http://ase.arc.nasa.gov/grosu/phd-thesis.ps`.

[23] Grigore Roşu. Abstract semantics for module composition. Technical Report CSE2000–0653, University of California at San Diego, May 2000.

[24] Grigore Roşu and Joseph Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 252–267. Springer, 2000. Papers from First Order Theorem Proving '98 (FTP'98).

[25] Andrzej Tarlecki. Moving between logical systems. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specification*, volume 1130 of *Lecture Notes in Computer Science*, pages 478–502. Springer, 1996.

[26] William Tracz. LILEANNA: a parameterized programming language. In *Proceedings, Second International Workshop on Software Reuse*, pages 66–78, March 1993. Lucca, Italy.