# UC San Diego
## Technical Reports

**Title**

Abstract Semantics for Module Composition


**Permalink**
https://escholarship.org/uc/item/1b09665j



**Author**
Rosu, Grigore



**Publication Date**
2000-05-08

Peer reviewed



eScholarship.org Powered by the California Digital Library
University of California

# Abstract Semantics for Module Composition

Grigore Roşu[1]
Department of Computer Science & Engineering,
University of California at San Diego
grosu@cs.ucsd.edu

## 1 Introduction

Technology is evolving unexpectedly fast. Software requirements tend to be exponentially higher every year and thus huge software systems are needed. Failure-safe systems are required not only in armies or governmental institutions, but also in many branches of industry. It is not a new thing that most failures in big systems are due either to flaws in requirements and incipient formal or informal specifications, or to exceptions which are not treated in implementations. In this light, good specification languages and automatic code generators seem to be indispensable, and modularization is becoming a crucial methodology.

Programmers and software engineers agree in unanimity that a useful characteristic of the programming languages they use for implementations (C++, Java, etc.) is their support for both *public* and *private* features (types, functions). The public features are often called *interfaces*. The private part is not visible outside the module (class, package) that declares it, but it can be used internally to define the visible part. Such distinction helps software engineers abstract their work and ignore details which are a main source of confusion and errors.

We claim that the distinction between private and public features might also be a desirable characteristic of formal specifications, not only from the practical point of view, because of the increased level of abstraction, but also because of at least two important theoretical reasons. One is the possibility to specify finitely some theories which do not admit finite standard presentations. For example, Bergstra and Tucker [2] showed that any recursive $\Sigma$-algebra can be specified as the $\Sigma$-restriction of an initial $\Sigma'$-algebra presented by a finite number of $\Sigma'$-equations, for some $\Sigma \subseteq \Sigma'$ (see [15] for a summary on this subject). The other theoretical reason is that it allows the user to specify behavioral properties of systems, in the sense that every behavioral specification [8, 18, 9] is equivalent to hiding some operators (making them private) in a usual specification [9].

Inspired by Goguen and Tracz's work [10], we introduce the notion of *module specification* as a generalization of the standard specification, having both public (or visible) and private features, and then we explore their properties at an abstract level, categorical, to make sure that our results are general enough to include the different concrete examples we know. To formalize the notion of "logical system" we use *institutions*, an intensively used abstract concept introduced by Goguen and Burstall [7]. The institution in which we work (we call it the *working institution*) is enriched with *inclusions* (of signatures) formulated in a categorical setting, an abstraction of the natural notion of inclusion of signatures from particular logics. A module specification in such an institution is a triple $(\varphi, \Sigma, A)$, where $\varphi$ is a subsignature of $\Sigma$, called the *visible signature* (or the public signature), and $A$ is a set of $\Sigma$-sentences; $\Sigma$ is called the *working signature* and it stands for both the public and the private symbols of that module. The *visible theorems* (or the visible consequences) of a module $(\varphi, \Sigma, A)$ are exactly the $\varphi$-sentences satisfied by $A$ over $\Sigma$, and a model of that module is a $\varphi$-model of its visible consequences.

The first interesting result in the paper is that the category of module specifications is equivalent to the category of theories. This does not surprise; informally, it says that module specifications represent a way to specify more elegantly some theories.

Five basic operations on module specifications are explored: renaming, hiding, enriching, aggregation and parameterization. An internal property of the module, called *conservativeness*, seems to have a decisive role in giving semantics for module composition. A module $(\varphi, \Sigma, A)$ is conservative iff every $\varphi$-model of its visible theorems can be extended to a $\Sigma$-model of $A$. We show that under conservatism, many of the

[1] Fundamentals of Computer Science, Faculty of Mathematics, University of Bucharest, Romania.

standard specifications' properties also hold for modules. For example, Theorem 78 says that the module given by an instantiation of a parameterized module is a pushout, as long as both the parameterized module and the actual parameter module are conservative.

Testing the conservatism of a a module $(\varphi, \Sigma, A)$ is a difficult task and it is dependent on the underlying logic. In many-sorted equational logics, for example, the technique consists in enriching a $\varphi$-algebra with some new carriers for the private sorts (in $\Sigma - \varphi$), and also with some new private operations, and then to show that the new $\Sigma$-algebra verifies the axioms in $A$. Of course, the fewer private features are added, the easier the job of testing the conservatism is. For this reason, we prefer to reduce the conservatism of a module with visible signature $\psi$ and working signature $\Sigma$, to the conservatism of other two modules: one of them with visible signature $\psi$ and working signature $\varphi$, for some $\psi \hookrightarrow \varphi \hookrightarrow \Sigma$, and the other one with visible signature $\varphi$ and working signature $\Sigma$ (see Propositions 59 and 64).

We consider the work in this paper as a bridge between the work by Goguen and Tracz [10] on implementation oriented semantics for module composition and the work by Diaconescu, Goguen and Stefaneas [6] on semantics for composition of specifications and theories.

# 2 Preliminaries

This section presents notations, definitions and properties useful later in the paper. First, some basic notions of category theory are exposed; then we introduce the concept of *inclusions* in a categorical setting, and then we remind the reader the basics of institutions, an abstract concept introduced by Goguen and Burstall to formalize the notion of "logical system".

## 2.1 Category Theory

The reader is supposed to be acquainted with some basic notions of category theory, such as categories and subcategories, products and coproducts, pullbacks and pushouts, functors and adjoints; we find [13] and [11] very good references on category theory.

$|\mathcal{C}|$ denotes the class of objects of a category $\mathcal{C}$, and for any two objects $A, B$ in $\mathcal{C}$, $\mathcal{C}(A, B)$ denotes the set of morphisms from $A$ to $B$. We write the composition of morphisms in diagrammatic order, that is $f; g\colon A \to C$ is the composition of $f\colon A \to B$ with $g\colon B \to C$. Let **Cat** denote the category of categories, having small categories as objects and functors as morphisms. A functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{D}$ is full (faithful) if its hom-set restriction $\mathcal{F}\colon \mathcal{C}(A, B) \to \mathcal{D}(\mathcal{F}(A), \mathcal{F}(B))$ is a surjective (injective) function for any objects $A, B$ in $\mathcal{C}$. The functor $\mathcal{F}$ is said to be *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 $\mathcal{C}$ is called *skeletal* if isomorphic objects are identical. A *skeleton* of a category $\mathcal{C}$ is a maximal full skeletal subcategory of $\mathcal{C}$; it can be readily seen that any two skeletons of a category are isomorphic in **Cat**. A category $\mathcal{C}$ is said to be equivalent to a category $\mathcal{D}$ if and only $\mathcal{C}$ and $\mathcal{D}$ have isomorphic skeletons. The following result is used in section 3 to show that the category of module specifications is equivalent to the category of theories in any institution:

**Proposition 1** Categories $\mathcal{C}$ and $\mathcal{D}$ are equivalent if and only if there exists a functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{D}$ which is full, faithful and dense. $\Box$

A very special kind of pullbacks are the pullbacks in **Cat**. They have the following important property:

**Proposition 2** If the 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 $C_2 \in |\mathcal{C}_2|$ 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 $\mathcal{F}_2(P) = C_2$. $\Box$

## 2.2 Inclusions

It is well-known that a small category can be associated to any partially ordered set: there exists exactly one object $A$ for each element $a$ in that set and there exists a morphism from $A$ to $B$, written $A \hookrightarrow B$, if and only if $a \leq b$. Furthermore, there is a bijection between partially ordered sets and small categories in

which there is at most one morphism from $A$ to $B$ for every objects $A$ and $B$ (partiality), and if there is a morphism from $A$ to $B$ and a morphism from $B$ to $A$ then $A = B$ (anti-symmetry). The correspondents of infimum and supremum are the product and the coproduct, respectively. Generalizing all these to categories which are not required to be small, we get:

**Definition 3** A category $\mathcal{I}$ is called a **category of inclusions** if and only if

- $\mathcal{I}(A, B)$ has at most one element, and
- $\mathcal{I}(A, B) \neq \emptyset$ and $\mathcal{I}(B, A) \neq \emptyset$ implies $A = B$.

for every pair of objects $A$ and $B$. If $\mathcal{I}(A, B) \neq \emptyset$ then let $A \hookrightarrow B$ denote the unique morphism in $\mathcal{I}(A, B)$. It is called an **inclusion** and $A$ is called a **subobject** of $B$. We say that $\mathcal{I}$ **has (finite) intersections** iff $\mathcal{I}$ has (finite) products and we say that $\mathcal{I}$ **has (finite) unions** iff $\mathcal{I}$ has (finite) coproducts. For every pair of objects $A$, $B$, let $A \cap B$ denote their product (also called their intersection) and let $A \cup B$ denote their coproduct (also called their union). □

A small category with finite intersections and finite unions corresponds to nothing else than a lattice. Consequently, many properties of lattices hold in categories with inclusions. The following are only a few:

**Fact 4** For any category of inclusions $\mathcal{I}$ and any objects $A$, $B$ and $C$ (assume that $\mathcal{I}$ has finite intersections and/or finite unions whenever $\cap/\cup$ appear),

1. $A \hookrightarrow A \cup B$ and $A \cap B \hookrightarrow A$,
2. $A \hookrightarrow B$ implies $A \cup B = B$ and $A \cap B = A$,
3. $A \cap (A \cup B) = A \cup (A \cap B) = A$,
4. $A \hookrightarrow B$ implies $A \cup C \hookrightarrow B \cup C$ and $A \cap C \hookrightarrow B \cap C$,
5. The union and intersection are commutative, associative and idempotent,
6. $(A \cap B) \cup (A \cap C) \hookrightarrow A \cap (B \cup C)$,
7. $A \cup (B \cap C) \hookrightarrow (A \cup B) \cap (A \cup C)$,

□

Since the union and intersection are associative, we take the liberty to use the natural notation $\bigcup_{j=1}^{n} A_j$ for the union of $n$ objects $A_1, ..., A_n$.

**Fact 5** The following are equivalent:

1. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ for all $A, B, C \in |\mathcal{I}|$,
2. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ for all $A, B, C \in |\mathcal{I}|$.

□

**Definition 6** $\mathcal{I}$ is **distributive** iff the equalities in Fact 5 hold. □

The notion of inclusion we introduced above is similar to the one of (weak) inclusion systems (see [6, 12, 4, 3] and also [17]) except that the factorization property is no longer required. In the present paper, we do not need the whole technical engine provided by (weak) inclusion systems.

The potential value of inclusion systems was suggested in [7], and a first definition was given in [6] in the context of modularization for standard specifications. The papers [12] and [4, 3] further simplify and generalize inclusion systems. Inclusion systems are an alternative of factorization systems (e.g., see [11, 16]). They can be preferred because the proofs tend to be smoother than using factorization systems, still having the same power of expressiveness.

**Definition 7** A category of inclusions $\mathcal{I}$ which is a broad subcategory[2] of $\mathcal{C}$ is called a **subcategory of inclusions** of $\mathcal{C}$ (alternatively, we can say that $\mathcal{C}$ **has inclusions** $\mathcal{I}$). $\mathcal{I}$ is a subcategory of **strong inclusions** of $\mathcal{C}$ (or $\mathcal{C}$ **has strong inclusions** $\mathcal{I}$) iff $\mathcal{I}$ is a subcategory of inclusions of $\mathcal{C}$, $\mathcal{I}$ has finite intersections and unions, and for every pair of objects $A$, $B$, their union in $\mathcal{I}$ is a pushout in $\mathcal{C}$ of their intersection in $\mathcal{I}$. $\mathcal{C}$ is **$\mathcal{I}$-distributive** iff $\mathcal{I}$ is distributive. □

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

**Fact 8** In any category $\mathcal{C}$ with strong inclusions $\mathcal{I}$,

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

□

The next definition introduces the notion of union of morphisms in a category with inclusions:

**Definition 9** If $\mathcal{C}$ is a category with inclusions $\mathcal{I}$ then the morphisms $h_1\colon A_1 \to B_1, ..., h_n\colon A_n \to B_n$ in $\mathcal{C}$ **admit unions** iff there are some morphisms $h\colon \bigcup_{j=1}^n A_j \to \bigcup_{j=1}^n B_j$ such that

$$(A_i \hookrightarrow \bigcup_{j=1}^n A_j); h = h_i; (B_i \hookrightarrow \bigcup_{j=1}^n B_j)$$

for each $i \in 1..n$. If the morphism $h$ above is unique then we say that $h_1, ..., h_n$ **admit union**, and we let $\bigcup_{j=1}^n h_j$ denote the unique morphism. □

**Fact 10** If $\mathcal{I}$ is a subcategory of strong inclusions then $h$ in the definition above is unique. □

**Definition 11** If $\mathcal{C}$ has an initial object $\emptyset$ then $A_1, ..., A_n$ are **disjoint w.r.t.** $\emptyset$ iff $A_i \cap A_j = \emptyset$ for all $i \neq j \in 1..n$. We say that $A_1, ..., A_n$ are **disjoint** whenever $\emptyset$ can be unambiguously inferred from the context. □

The following proposition shows conditions under which unions of morphisms exist:

**Proposition 12** Let $\mathcal{C}$ be a category having strong inclusions $\mathcal{I}$ and let $h_1\colon A_1 \to B_1, ..., h_n\colon A_n \to B_n$ be morphisms in $\mathcal{C}$. Then $h_1, ..., h_n$ admit union whenever at least one of the following holds:

1. $(A_i \cap A_j \hookrightarrow A_i); h_i; (B_i \hookrightarrow B_i \cup B_j) = (A_i \cap A_j \hookrightarrow A_j); h_j; (B_j \hookrightarrow B_i \cup B_j)$ for every $i, j \in 1..n$, or
2. $\mathcal{C}$ has an initial object $\emptyset$ and $A_1, ..., A_n$ are disjoint.

□

The following definition plays an important technical role in the present paper. It formalizes categorically a crucial property of signatures:

**Definition 13** A category with inclusions has **pushouts which preserve inclusions** iff for any pair of arrows $(A \hookrightarrow B, A \to A')$ there are some pushouts of the form $(A' \hookrightarrow B', B \to B')$. □

**Example 14** Sets: The category of sets has pushouts which preserve inclusions. Indeed, let $A \hookrightarrow B$ be an inclusion and $h\colon A \to A'$ be a function; then let $B'$ denote the set $A' \coprod (B - A)$, where $\coprod$ stands for the disjoint union. There are many different ways in which the disjoint sum can be done; however, we consider that it is done by providing copies of the elements in $B - A$ which are not in $A'$, such that $A'$ is a pure subset of $A' \coprod (B - A)$ (let $\overline{b}$ be the corresponding copy of $b$ in $B - A$). Let $h'\colon B \to B'$ be the function defined as $h'(b) = h(b)$ for $b \in A$ and $h'(b) = \overline{b}$ for $b \in B - A$. Then the pair of functions $A' \hookrightarrow B'$ and $h'\colon B \to B'$ is a pushout preserving the inclusion $A \hookrightarrow B$. Notice that more pushouts that preserve the inclusion $A \hookrightarrow B$ exist.. □

**Example 15** Signatures: The category of signatures also has pushouts which preserve inclusions. Consider a signature inclusion $(R, \varphi) \hookrightarrow (S, \Sigma)$ and a signature morphism $(f, g)\colon (R, \varphi) \to (R', \varphi')$. Let $(R' \hookrightarrow S', f'\colon S \to S')$ be a pushout in the category of sets (as in the example above) of the pair $(R \hookrightarrow S, f\colon R \to R')$, and let $\Sigma'$ be $\varphi' \coprod f'(\Sigma - \varphi)$, where the disjoint union is done by eventually renaming the elements in $f'(\Sigma - \varphi) = \{\sigma\colon f'(w) \to f'(s) \mid \sigma\colon w \to s \in \Sigma - \varphi\}$. It is left to the reader to check that $((R', \varphi') \hookrightarrow (S', \Sigma'), (f', g')\colon (S, \Sigma) \hookrightarrow (S', \Sigma'))$ is a pushout, where $g'(\sigma\colon w \to s) = \overline{\sigma}\colon f'(w) \to f'(s)$. □

We consider that any specification language which is designed to offer support for modularization, including renaming and parameterizations, should offer a way to do pushouts which preserve inclusions at the signatures level. This is in our opinion the unique consistent way to extend the renaming of a subsignature $\varphi$ of $\Sigma$ to the whole signature $\Sigma$. Even if OBJ does not offer full support for our modularization approach[3], it is able to do pushouts preserving the inclusions; the following example shows how it does that:

**Example 16** Let us consider the following OBJ program which defines a signature[4] `PHI` with one sort $S$ and one unary operation $a\colon S \to S$, an extension `SIGMA`[5] of `PHI` which adds a new sort $S'$ and a new operation $a\colon S \to S'$, and another signature `PHI'` which defines another sort $S'$ and an operation $a\colon S' \to S'$. To get the pushout, we use the instantiation operation provided by OBJ's module system.

```
th PHI is
  sort S .
  op a : S -> S .
endth

th SIGMA[X :: PHI] is
  sort S' .
  op a : S -> S' .
endth

th PHI' is
  sort S' .
  op a : S' -> S' .
endth

show SIGMA[PHI'] .
```

When OBJ calculates the instantiation `SIGMA[PHI']`, a default view (morphism) from `PHI` to `PHI'` is considered, which takes $S$ to $S'$ and $a\colon S \to S$ to $a\colon S' \to S'$. The output is:

```
theory SIGMA[view to PHI' is sort S to S' . op a(v0) to (a(v0)).PHI' .
     endv] is
  protecting PHI' .
  sorts S'.(SIGMA[view to PHI' is sort S to S' . op a(v0) to (a(v0)).
    PHI' . endv]) S'.PHI' Bool .
  op a : S'.PHI' -> S'.PHI' .
  op a : S'.PHI' -> S'.(SIGMA[view to PHI' is sort S to S' . op a(v0)
     to (a(v0)).PHI' . endv]) .
endth
```

Observe that OBJ automatically renamed the sort $S'$ from `SIGMA` in order to avoid the conflict with $S'$ from `PHI'`. It is straightforward that the new signature is a pushout which preserves the inclusion. □

The problem with pushouts which preserve inclusions is that there can be more pushouts preserving the inclusion. For example, the operation `a` from `S'.PHI'` to `S'.(SIGMA[view to PHI' is sort S to S' . op a(v0) to (a(v0)).PHI' . endv])` in the theory above, could have been very well renamed as `b` with the same source and target as `a`, and the new signature is still a pushout which preserves the inclusion.

**Notation 17** We do neither impose any particular algorithm for doing such pushouts in particular examples, nor ask any additional property they have to respect; but for notational sake, we assume a *fixed* pushout which preserves the inclusion for any diagram $(A \hookrightarrow B, h\colon A \to A')$. Let $(A' \hookrightarrow B_h, h_B\colon B \to B_h)$ denote this special pushout. □

[3]Everything declared into a module is public in OBJ, unlike in our approach where modules may have some private sorts and operations.

[4]We use theories to embed signatures in order to take advantage of the OBJ's module system operations.

[5]We use a parameterized theory for this extension; actually, the parameterized theories are stronger than this example shows.

**Open Problem:** We would like to find an algorithm able to calculate the pushout above for standard signatures, such that to be closed to horizontal and/or vertical composition of pushouts, that is, if $(\varphi \hookrightarrow \Sigma, h\colon \varphi \to \varphi')$ is a pair of signature morphisms then $\Sigma'_{(h_\Sigma)} = \Sigma'_h$ for each inclusion of signatures $\Sigma \hookrightarrow \Sigma'$, and/or respectively $(\Sigma_h)_g = \Sigma_{h;g}$ for each signature morphism $g$ having the source $\varphi'$.

**Definition 18** A functor between two categories with inclusions **preserves inclusions** iff it takes inclusions in the source category to inclusions in the target category. □

## 2.3 Institutions

The concept of institution was introduced by Goguen and Burstall [7] to formalize the abstract notion of "logical system". Having as basic idea the Tarski's classic semantic definition of truth, institutions have the possibility of translating sentences and models along signature morphisms, with respect to an axiom called the satisfaction condition, which says that *truth is invariant under change of notation.*

**Definition 19** 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 $\Sigma$-sentences, a functor $\mathbf{Mod}\colon \mathbf{Sign} \to \mathbf{Cat}^{op}$ giving for each signature $\Sigma$ a category of $\Sigma$-models, and a $\Sigma$-indexed relation $\models = \{\models_\Sigma \mid \Sigma \in \mathbf{Sign}\}$, where $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$, such that for each signature morphism $h\colon \Sigma \to \Sigma'$, the following **Satisfaction Condition**

$$m' \models_{\Sigma'} \mathbf{Sen}(h)(a) \ \text{ iff } \ \mathbf{Mod}(h)(m') \models_\Sigma a$$

holds for each $m' \in |\mathbf{Mod}(\Sigma')|$ and each $a \in \mathbf{Sen}(\Sigma)$. □

We sometimes write only $h$ instead of $\mathbf{Sen}(h)$ and $_\!\restriction_h$ instead of $\mathbf{Mod}(h)$; the functor $_\!\restriction_h$ is called the reduct functor associated to $h$. With these notations, the satisfaction condition becomes $m' \models_{\Sigma'} h(a)$ iff $m'\!\restriction_h \models_\Sigma a$. The satisfaction notation, $\models_\Sigma$, is also used for a set of sentences in the right side, that is we write $m \models_\Sigma A$ for $A$ a set of $\Sigma$-sentences, meaning that $m$ satisfies each sentence in $A$. Moreover, we extend this notation for sets of sentences on both sides: $A \models_\Sigma A'$ means that $m \models_\Sigma A'$ for any $\Sigma$-model $m$ with $m \models_\Sigma A$. We forget the subscript $\Sigma$ in $\models_\Sigma$ whenever it can be inferred unambiguously from the context. The *closure* of a set of $\Sigma$-sentences $A$ is the set denoted $A^\bullet$ which contains all $a$ in $\mathbf{Sen}(\Sigma)$ such that $A \models_\Sigma a$. The sentences in $A^\bullet$ are often called the theorems of $A$. Obviously the closure operation is a closure operator, that is it is extensive, monotonic and idempotent.

**Closure Lemma:** For any signature morphism $h\colon \Sigma \to \Sigma'$ and any set of $\Sigma$-sentences $A$, $h(A^\bullet) \subseteq h(A)^\bullet$. □

**Fact 20** For any sets of $\Sigma$-sentences $A$ and $A'$, and any signature morphism $h\colon \Sigma \to \Sigma'$,

- $h(A^\bullet)^\bullet = h(A)^\bullet$, and
- $(A^\bullet \cup A')^\bullet = (A \cup A')^\bullet$.

□

**Definition 21** A **specification** or **presentation** is a pair $(\Sigma, A)$ where $\Sigma$ is a signature and $A$ is a set of $\Sigma$-sentences. A **specification morphism** from $(\Sigma, A)$ to $(\Sigma', A')$ 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** $(\Sigma, A)$ is a specification with $A = A^\bullet$; the full subcategory of theories in **Spec** is denoted **Th**. □

It can be readily seen that the categories **Th** and **Spec** are equivalent. The equivalence functor is just the inclusion of categories $\mathcal{U}_s\colon \mathbf{Th} \to \mathbf{Spec}$. It 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 identity on morphisms; note that $\mathcal{F}_s$ is also a right adjoint of $\mathcal{U}_s$, so **Th** is a reflective and coreflective subcategory of **Spec**.

It is known (see [7]) that **Th** is cocomplete whenever **Sign** is cocomplete; in particular, **Th** has pushouts whenever **Sign** has pushouts. We remind the reader how pushouts are built in **Th**:

**Proposition 22** If $h_1: (\Sigma, A) \to (\Sigma_1, A_1)$ and $h_2: (\Sigma, A) \to (\Sigma_2, A_2)$ are two theory morphisms and $(h'_1: \Sigma_1 \to \Sigma', h'_2: \Sigma_2 \to \Sigma')$ a pushout of $(h_1, h_2)$ in **Sign**, then $(h'_1: (\Sigma_1, A_1) \to (\Sigma', A'), h'_2: (\Sigma_2, A_2) \to (\Sigma', A'))$ is a pushout in **Th** of $h_1$ and $h_2$, where $A' = (h'_1(A_1) \cup h'_2(A_2))^\bullet$. □

**Definition 23** A theory morphism $h: (\Sigma, A) \to (\Sigma', A')$ is **conservative** if for any $(\Sigma, A)$-model $m$ there are some $(\Sigma', A')$-models $m'$ such that $m'\lceil_h = m$. The signature morphism $h: \Sigma \to \Sigma'$ is conservative if it is conservative as a morphism of void theories, i.e. $h: (\Sigma, \emptyset^\bullet) \to (\Sigma', \emptyset'^\bullet)$. □

**Proposition 24** If $h: \Sigma \to \Sigma'$ and $a \in \mathbf{Sen}(\Sigma)$ and $A \subseteq \mathbf{Sen}(\Sigma)$ then

1. $A \models_\Sigma a$ implies $h(A) \models_{\Sigma'} h(a)$.
2. If $h$ is conservative then $A \models_\Sigma a$ iff $h(A) \models_{\Sigma'} h(a)$.

**Proof:** The first assertion follows directly from the closure lemma, and for the second let $m \models_\Sigma A$. Because $h$ is conservative, there are some $\Sigma'$-models $m'$ such that $m'\lceil_h = m$; thus $m'\lceil_h \models_\Sigma A$, that is $m' \models_{\Sigma'} h(A)$. Therefore $m' \models_{\Sigma'} h(a)$, and so $m \models_\Sigma a$. □

**Definition 25** An institution **has (strong) inclusions**, or it is an institution **with (strong) inclusions** iff its category of signatures has inclusions and **Sen** preserves them. It is **distributive** iff its category of signatures is distributive. An institution is called **semiexact** if the functor $\mathbf{Mod}: \mathbf{Sign} \to \mathbf{Cat}^{op}$ preserves the pushouts[6], i.e. it takes pushouts in **Sign** to pullbacks in **Cat**. □

The term semiexactness was introduced by Diaconescu, Goguen and Stefaneas [6] as a weakening of *exactness*. The property of exactness, which says that **Mod** preserves the general colimits, seems to have first appeared in [19] and then used by Tarlecki [20] on abstract algebraic institutions and by Meseguer [14] on categorical logic. Many sorted logics tend to be exact, but their unsorted variants tend to be only semiexact.

**Fact 26** If $A$ and $A'$ are sets of $\Sigma$ and $\Sigma'$-sentences, respectively, then $(A^\bullet \cup A'^\bullet)^\bullet = (A \cup A')^\bullet$, where the outermost closures are done over $\Sigma \cup \Sigma'$-sentences. □

The category of theories, **Th**, tends to have many of the properties of **Sign**. One of the most important properties was pointed above, and it says that **Th** is cocomplete as long as **Sign** is cocomplete. In our framework with inclusions, we get:

**Proposition 27** In any institution with (strong) inclusions,

1. **Th** has (strong) inclusions.
2. **Th** has pushouts which preserve inclusions whenever **Sign** has pushouts which preserve inclusions.

**Proof:**

1. The category of inclusions $\mathcal{I}_{\mathbf{Th}}$ in **Th** contains exactly the morphisms denoted $(\Sigma, A) \hookrightarrow (\Sigma', A')$, where $\Sigma \hookrightarrow \Sigma'$ is an inclusion in **Sign** and $A \subseteq A'$ as sets of $\Sigma'$-sentences. It is easy to check that $\mathcal{I}_{\mathbf{Th}}$ is a partial order and that it has the same objects as **Th**, i.e. it verifies the first conditions in Definition 3. Define the union of two theories $(\Sigma, A)$ and $(\Sigma', A')$, by $(\Sigma, A) \cup (\Sigma', A') = (\Sigma \cup \Sigma', (A \cup A')^\bullet)$ where the closure is done over $\Sigma \cup \Sigma'$-sentences, and their intersection by $(\Sigma, A) \cap (\Sigma', A') = (\Sigma \cap \Sigma', A \cap A')$. The correctness of these definitions is obvious, and the strongness condition is also respected by $\mathcal{I}_{\mathbf{Th}}$ because of the construction of pushouts in **Th** (see Proposition 22).
2. Let $(\Sigma, A) \hookrightarrow (\Sigma_1, A_1)$ be an inclusion in **Th** and $h: (\Sigma, A) \to (\Sigma_2, A_2)$ be a morphism in **Th**. Take $(\Sigma_2 \hookrightarrow \Sigma', h_{\Sigma'}: \Sigma_1 \to \Sigma')$ a pushout of $(\Sigma \hookrightarrow \Sigma_1, h: \Sigma \to \Sigma_2)$ in **Sign** which preserves the inclusion. We claim that $((\Sigma_2, A_2) \hookrightarrow (\Sigma', A'), h_{\Sigma'}: (\Sigma_1, A_1) \to (\Sigma', A'))$ is the desired pushout in **Th**, where $A' = (A_2 \cup h_{\Sigma'}(A_1))^\bullet$; it follows immediately from Proposition 22.

□

Sometimes, we say **theory extension** instead of inclusion of theories .

[6] Actually, we are interested only in pushouts of inclusions, but we try to avoid introducing a new concept.

# 3 Module Specifications

Information hiding is an important technique in modern computer science, both in programming and algebraic specifications. There are some interesting examples of $\varphi$-theories which cannot have finite $\varphi$-presentations, but they have finite $\Sigma$-presentations for a larger signature $\Sigma$, that is, the restriction of the finite $\Sigma$-presented theory to $\varphi$ is exactly the infinite $\varphi$-presented theory (see Example 30).

Our definition of module specifications extend the usual algebraic specifications, letting them to have visible (or public) features and also private features; the private part is useful to express the visible part. Only the visible consequences (theorems) of a module characterize the semantics of the module specification, and any implementation of the module has to respect them.

**Assumption:** From now on in the paper, suppose that we work in an institution with inclusions; we call it the **working institution**.

## 3.1 Visible Theorems

The visible consequences of a module play an important role in our approach.

**Definition 28** For $\imath\colon \varphi \hookrightarrow \Sigma$ an inclusion in **Sign** and $A$ a set of $\Sigma$-sentences, we let $Th^{\Sigma}_{\varphi}(A)$ denote the set $\imath^{-1}(A^{\bullet})$ of $\varphi$-sentences, and we call it the **$\varphi$-visible theorems**[7] of $A$. $\Box$

In other words, $Th^{\Sigma}_{\varphi}(A)$ contains all the sentences $a \in \mathbf{Sen}(\varphi)$ such that $A \models_{\Sigma} a$, i.e. it contains all the $\varphi$-sentences which are consequences of $A$. When $\imath$ is an identity, we simply get $Th^{\Sigma}_{\Sigma}(A) = A^{\bullet}$, that is the $\Sigma$-visible theorems of $A$ are exactly the theorems of $A$.

**Example 29** The following module has a private operation, `aux`, which is an auxiliary operation helping to define the main operation, `rev`, which reverses lists:

```
obj REV[X :: TRIV] is pr LIST[X] .
  op rev : List -> List .        *** public
  op aux : List List -> List .   *** private
  var E : Elt . vars L P : List .
  eq rev(L) = aux(L, nil) .
  eq aux(nil, P) = P .
  eq aux(E L, P) = aux(L, E P) .
endo
```

Then the following are only a few visible theorems:

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

The first property is immediate, but the second one is more complicated. We advice the curios reader first to prove the lemma $(\forall L_1, L_2\colon\ List)\ aux(L_1, L_2) = (aux(L_1, nil)\ L_2)$ by induction on the length of $L_1$. The third follows from the first two by induction on $L$, and the fourth follows from the first three also by induction on $L$.

A natural question here is why to define `rev` so complicatedly, instead of defining it much easier as in the following module with no private operations:

```
obj REV[X :: TRIV] is pr LIST[X] .
  op rev : List -> List .
  var E : Elt .  var L : List .
  eq rev(nil) = nil .
  eq rev(E L) = rev(L) E .
endo
```

[7] Sometimes, we call the set $\imath^{-1}(A^{\bullet})$ the **$\varphi$-visible theorems over** $\Sigma$, if $\Sigma$ is not clear from the context.

The answer is: because of efficiency reasons, the first module using tail recursion. A list of 25 elements is reversed more than 5 times faster using the first module than using the second one (under both OBJ3 and CafeOBJ). Thus, the first module `REV` should be viewed as a refinement of the second. □

**Example 30** This example is inspired from [8]. It shows how stacks can be specified with the help of arrays. The sorts `Stack` and `NeStack` together with the operations `empty`, `top`, `push` and `pop` are public and the other operations which are related to arrays are private:

```
th STACK[X :: TRIV] is pr NAT .
*** the public signature
  sorts Stack NeStack .
  subsort NeStack < Stack .
  op empty : -> Stack .
  op top   : NeStack -> Elt .
  op push  : Elt Stack -> NeStack .
  op pop   : NeStack -> Stack .
*** the private signature and equations of arrays:
  sort Array .
  op nil : -> Array .
  op put : Elt NzNat Array -> Array .
  op _[_] : Array NzNat -> Elt .
  vars J K : NzNat . var E : Elt . var A : Array .
  cq put(E,J,A)[K] = E if J == K .
  cq put(E,J,A)[K] = A[K] if J =/= K .
*** the private constructors of the sorts Stack and NeStack:
  op _||_ : Nat Array -> Stack .
  op _||_ : NzNat Array -> NeStack .
*** equations defining the public operations:
  var I : Nat .
  eq empty = 0 || nil .
  eq top(J || A) = A[J] .
  eq push(E, I || A) = s I || put(E, s I, A) .
  eq pop(s I || A) = I || A .
endth
```

Then the following expressions are visible theorems:

- $(\forall E : Elt, S : Stack)\ top(push(E, S)) = E$,
- $(\forall E : Elt, S : NeStack)\ top(pop(push(E, S))) = top(S)$,

  $\vdots$
- $(\forall E_1, \ldots, E_n : Elt, S : NeStack)\ top(pop(\ldots pop(push(E_1, \ldots, push(E_n, S))))) = top(S)$.

  $\vdots$

where the number of *pop*'s is equal to the number of *push*'s is equal to $n$ in the last expression. To prove them, one needs to consider that the two operations `_||_` are constructors for the sorts `NeStack` and `Stack`.

Notice that the expression $(\forall E : Elt, S : Stack)\ pop(push(E, S)) = S$ is not a visible theorem. Therefore, the module above does not refine the following well-known `Stack` module (with no private symbols):

```
th STACK[X :: TRIV] is
  sorts Stack NeStack .
  subsorts NeStack < Stack .
  op empty : -> Stack .
  op top : NeStack -> Elt .
  op push : Elt Stack -> NeStack .
  op pop : NeStack -> Stack .
```

```
  var E : Elt . var S : Stack .
  eq top(push(E, S)) = E .
  eq pop(push(E, S)) = S .
endth
```

Actually, the expression $(\forall E : Elt, S : Stack)\ pop(push(E,S)) = S$ is not desired as a property of stacks at all. This is because almost no real implementation (i.e., model) of stacks respects it, including the one with pointer in array following the idea of the first module in this example. We claim that the visible theorems above together with many similar others having `top` as uppermost operation[8] are *exactly* the desired properties of every implementation of stacks. The internal data structure is most often hidden in implementations, the only way to "observe" a stack being to apply a `top` after a chain of `push`'s and/or `pop`'s. In this light, the first `STACK` module can be viewed as a *finite* specification of an infinite abstract data type. Notice that this thing would have not been possible without hiding some operations, making them private.

A different technique to specify finitely an infinite ADT is based on *behavioral specifications*, in which the models (or hidden algebras) are allowed to satisfy behaviorally the properties, i.e., with respect to observations (attributes) following actions (methods). A behavioral specification of stacks would be similar to the second `STACK` module in this example, except that the last equation is behavioral. We are not going to talk about behavioral specifications and hidden algebra, but we refer the interested reader to [8]). □

Truth is preserved under extensions of signatures in any logical system. The following proposition says that this also holds in any institution with inclusions.

**Proposition 31** If $\varphi \hookrightarrow \Sigma$ and $a \in \mathbf{Sen}(\varphi)$ and $A \subseteq \mathbf{Sen}(\varphi)$ then

1. $A \models_\varphi a$ implies $A \models_\Sigma a$.
2. $A \models_\varphi a$ iff $A \models_\Sigma a$ if the inclusion $\varphi \hookrightarrow \Sigma$ is conservative.

**Proof:** It follows immediately from Proposition 24 where $h$ is the inclusion $\varphi \hookrightarrow \Sigma$. □

The following fact presents properties of visible theorems; they will be often tacitly used later in the paper.

**Fact 32** Suppose the inclusions $\psi \hookrightarrow \varphi \hookrightarrow \Sigma$ in **Sign** and $A \subseteq A' \subseteq \mathbf{Sen}(\Sigma)$ and $B \subseteq \mathbf{Sen}(\varphi)$. Then

1. $B \subseteq Th_\varphi^\Sigma(B)$.
2. $Th_\psi^\varphi(B) \subseteq Th_\psi^\Sigma(B)$.
3. $Th_\psi^\varphi(B) = Th_\psi^\Sigma(B)$ if the inclusion $\varphi \hookrightarrow \Sigma$ is conservative.
4. $Th_\psi^\Sigma(A) \subseteq Th_\varphi^\Sigma(A)$.
5. $Th_\varphi^\Sigma(A) \subseteq Th_\varphi^\Sigma(A')$.
6. $Th_\psi^\Sigma(A) \subseteq Th_\varphi^\Sigma(Th_\psi^\Sigma(A))$.
7. $Th_\psi^\Sigma(Th_\varphi^\Sigma(A)) \subseteq Th_\psi^\Sigma(A)$.
8. $Th_\varphi^\Sigma(Th_\varphi^\Sigma(A)) = Th_\varphi^\Sigma(A)$.
9. $Th_\varphi^\varphi(Th_\varphi^\Sigma(A)) = Th_\varphi^\Sigma(A)$.
10. $Th_\psi^\varphi(Th_\varphi^\Sigma(A)) = Th_\psi^\Sigma(A)$.

**Proof:** Let $\imath' : \psi \to \varphi$ and $\imath : \varphi \to \Sigma$ be the two inclusions.

1. Straightforward, because if $b \in B$ then $B \models_\Sigma b$, that is $b \in Th_\varphi^\Sigma(B)$.
2. This is because of 1. in Proposition 31.
3. It is exactly 2. in Proposition 31.
4. Straightforward, because **Sen** is a morphism of inclusion systems, and so, $a$ is in $\mathbf{Sen}(\varphi)$ whenever $a$ is in $\mathbf{Sen}(\psi)$.
5. It is equivalent to $\imath^{-1}(A^\bullet) \subseteq \imath^{-1}(A'^\bullet)$, which is true because $A^\bullet \subseteq A'^\bullet$.
6. It follows from 1. with $Th_\psi^\Sigma(A)$ instead of $B$.

[8]For example, $(\forall E_1, E_2, E_3 : Elt, S : NeStack)\ top(pop(pop(push(E_1, pop(push(E_2, push(E_3, S))))))) = top(S)$

7. It 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. It follows from 6. and 7., for $\psi = \varphi$.
9. From 1., we deduce that $Th^{\Sigma}_{\varphi}(A) \subseteq Th^{\varphi}_{\varphi}(Th^{\Sigma}_{\varphi}(A))$. On the other hand, $Th^{\varphi}_{\varphi}(Th^{\Sigma}_{\varphi}(A)) \subseteq Th^{\Sigma}_{\varphi}(Th^{\Sigma}_{\varphi}(A))$ by 2., and furthermore $Th^{\varphi}_{\varphi}(Th^{\Sigma}_{\varphi}(A)) \subseteq Th^{\Sigma}_{\varphi}(A)$ by 8..
10. It is equivalent to $\imath'^{-1}(\imath^{-1}(A^\bullet)) = (\imath';\imath)^{-1}(A^\bullet)$, which is true in any weak inclusion system, in particular in *Sets*.

□

The following lemma is a generalization of the Closure Lemma for an institution with inclusions.

**Lemma 33** **Generalization of the Closure Lemma:** If $\imath\colon \varphi \hookrightarrow \Sigma$ and $\imath'\colon \varphi' \hookrightarrow \Sigma'$ are two inclusions and $h\colon \varphi \to \varphi'$, $g\colon \Sigma \to \Sigma'$ are two morphisms such that $\imath; g = h; \imath'$, then $h(Th^{\Sigma}_{\varphi}(A)) \subseteq Th^{\Sigma'}_{\varphi'}(g(A))$.

**Proof:** Let $a$ be a $\varphi$-sentence in $Th^{\Sigma}_{\varphi}(A)$, that is $A \models_{\Sigma} a$. Then by Proposition 24, we get $g(A) \models_{\Sigma'} g(a)$. But $g(a) = h(a)$ because **Sen** preserves the inclusions, and so $h(a)$ belongs to $Th^{\Sigma'}_{\varphi'}(g(A))$. □

The standard Closure Lemma can be obtained from the previous lemma when $\imath$ and $\imath'$ are identities, that is, when there are no private features.

## 3.2 Motivation

We have already shown in Example 30 that the module specifications help us finitely specify infinite data structures. Moreover, Roşu and Goguen [18] showed that a (finite) equivalent module specification can be built for any (finite) behavioral specification, so in some way, the module specifications are more general than the behavioral specifications.

The next example presents a possible scenario in a software project in which hiding information prove to be an elegant solution to avoid capturing operation symbols:

**Example 34** Example 29 showed a refinement of the `REV` module, in which an auxiliary operation `aux` was introduced in order to transform the slow recursion in a more efficient tail recursion. It is often the case that some modules are real bottle-necks in big software projects, so their refinement can produce a serious improvement over the whole program. However, the refinement of a particular module should not affect the functionality of the other modules. Let us assume the following scenario in a software project.

- X writes the module `REV` using the inefficient recursion (the second `REV` module in Example 29).
- Many other programmers import the `REV` module. Among them, Y writes the code:

```
obj Y is pr REV[NAT] .
  op zip : List -> List .
  op aux : List List -> List .
  vars L P : List . vars N M : Nat .
  eq zip(L) = aux(L, rev(L)) .
  eq aux (nil, nil) = nil .
  eq aux (N L, M P) = N M aux(L, P) .
endo
```

  Y's intention is to zip a list with its reverse. For example, `zip(1 2 3 4 5)` gives `(1 5 2 4 3 3 4 2 5 1)`. Y's choice (very natural, anyway) is to define an auxiliary operation `aux` which zips two lists of the same length and then to define the `zip` operation using the auxiliary function as above.
- X receives many complains from other members in the team that the module `REV` is not efficient, so he decides to refine it using tail recursion, as in the first `REV` module in Example 29, making it five times faster.

If `aux` cannot be declared private in `REV`, then the functionality of the module `Y` is unexpectedly damaged. This is because `REV` captured an operation in `Y` which happened to have the same name, `aux`. The result of `zip(1 2 3 4 5)` is now `(5 4 3 2 1 5 4 3 2 1)`.

Overloading resolution might seem to solve this kind of problems, but unfortunately this is not the case as long as order sorted specifications and polymorphism are desired. For example, a supersort `List?` of `List` can be introduced in the module `Y` and `aux` can be declared as `aux : List List? -> List`, and still the same wrong result is obtained.

Consequently, a local refinement damaged the whole program instead of producing a global refinement as expected. If there is no mechanism to allow hiding operations, then discussion between programmers seems to be the only solution to avoid conflicts on refinements. □

## 3.3 Definition and Properties of Module Specifications

This subsection first introduces the very intuitive notion of module specification, and then shows that the associated category is equivalent to **Th**. Finally, the conservatism of a module, which says that the public features of that module together with the private ones give a conservative extension of the public features, is presented as a basic property of module specifications.

**Definition 35** A **module specification** is a triple $(\varphi, \Sigma, A)$, where $\varphi, \Sigma \in$ **Sign** such that $\varphi \hookrightarrow \Sigma$ and $A$ is a set of $\Sigma$-sentences. The signature $\varphi$ is called the **visible (or public) signature** and $\Sigma$ is called the **working signature**. A morphism $h\colon (\varphi, \Sigma, A) \to (\varphi', \Sigma', A')$ of module specifications is a signature morphism $h\colon \varphi \to \varphi'$ such that $h(Th^{\Sigma}_{\varphi}(A)) \subseteq Th^{\Sigma'}_{\varphi'}(A')$. □

**Notation 36** We let upper-case letters like $M, N, P, \ldots$ denote module specifications. For a module $M = (\varphi, \Sigma, A)$, let $Th(M)$ denote the set $Th^{\Sigma}_{\Sigma}(A)$ and call it the set of working theorems of $M$, and by $Vth(M)$ the set of $\varphi$-visible theorems of $A$, and call it the set of visible theorems of $M$. □

**Fact 37** The module specifications together with the morphisms of module specifications give a category. □

**Notation 38** Let **MSpec** denote the category of module specifications with morphisms of module specifications. □

**Definition 39** We define an application $\mathcal{M}$ from **Th** to **MSpec**, by $\mathcal{M}(\Sigma, A) = (\Sigma, \Sigma, A)$ for any theory $(\Sigma, A)$, and $\mathcal{M}(h) = h$ for any morphism of theories $h$. □

**Proposition 40** $\mathcal{M}\colon$ **Th** $\to$ **MSpec** is an equivalence of categories.

**Proof:** According to Proposition 1, it suffices to show that $\mathcal{M}$ is a full, faithful and dense functor. First, observe that $\mathcal{M}$ is well-defined; this is because for any morphism $h\colon (\Sigma, A) \to (\Sigma', A')$ in **Th** we have

$$\begin{array}{rcl} h(Th^{\Sigma}_{\Sigma}(A)) & = & h(A^{\bullet}) \\ & = & h(A) \\ & \subseteq & A'^{\bullet} \\ & = & Th^{\Sigma'}_{\Sigma'}(A') \end{array}$$

that is $\mathcal{M}(h)$ is a morphism in **MSpec**. It is left to the reader to check that $\mathcal{M}$ is a functor and that it is full and faithful.

Now, let $(\varphi, \Sigma, A)$ be a module specification and consider the theory $(\varphi, Th^{\Sigma}_{\varphi}(A))$. We claim that $(\varphi, \Sigma, A)$ and $\mathcal{M}(\varphi, Th^{\Sigma}_{\varphi}(A)) = (\varphi, \varphi, Th^{\Sigma}_{\varphi}(A))$ are isomorphic, and that the signature identity $1_{\varphi}$ is the desired isomorphism. Indeed, $1_{\varphi}\colon (\varphi, \Sigma, A) \to (\varphi, \varphi, Th^{\Sigma}_{\varphi}(A))$ is obviously a morphism of module specifications, and its inverse $1_{\varphi}\colon (\varphi, \varphi, Th^{\Sigma}_{\varphi}(A)) \to (\varphi, \Sigma, A)$ is also a morphism because $Th^{\varphi}_{\varphi}(Th^{\Sigma}_{\varphi}(A)) \subseteq Th^{\Sigma}_{\varphi}(A)$ (see Fact 32). □

**Corollary 41** $\mathcal{M}$ is part of an adjoint equivalence, where its left adjoint $\mathcal{T}\colon \mathbf{MSpec} \to \mathbf{Th}$ is defined as $\mathcal{T}(\varphi, \Sigma, A) = (\varphi, Th^{\Sigma}_{\varphi}(A))$ on objects and identity on morphisms; the unit of adjunction is the identity $1_{\varphi}\colon (\varphi, \Sigma, A) \to (\varphi, \varphi, Th^{\Sigma}_{\varphi}(A))$.

**Proof:** It follows from Theorem 1, pg. 91 in [13]. □

**Notation 42** Let $\mathcal{U}\colon \mathbf{MSpec} \to \mathbf{MSpec}$ denote the functor $\mathcal{T}; \mathcal{M}$, taking modules $(\varphi, \Sigma, A)$ to modules $(\varphi, \varphi, Th^{\Sigma}_{\varphi}(A))$. □

Thus the categories **Th** and **MSpec** are equivalent. Notice that $\mathcal{T}$ is also a right adjoint of $\mathcal{M}$. Therefore, **Th** can be (modulo an isomorphism) viewed as a reflective and coreflective subcategory of **MSpec**.

**Proposition 43** **MSpec** is cocomplete if **Sign** is cocomplete.

**Proof:** It is known that **Th** is cocomplete if **Sign** is cocomplete (see [7]); because **MSpec** and **Th** are equivalent, we deduce that **MSpec** is cocomplete, too. □

Moreover, since the two categories are equivalent, every "categorical property" (see [11]) of **Th** is a property of **MSpec**, too. In particular, the pushouts are preserved and reflected by the functors $\mathcal{M}$ and $\mathcal{T}$.

As we have already said, only the visible consequences of a module specification has to be respected by any implementation of a module:

**Definition 44** We say that a $\varphi$-model $m$ satisfies the module specification $M = (\varphi, \Sigma, A)$ if and only if $m \models_{\varphi} Vth(M)$, and write $m \models M$. □

**Proposition 45** Let $M = (\varphi, \Sigma, A)$ and $M' = (\varphi', \Sigma', A')$ be two module specifications, $h\colon M \to M'$ be a morphism of module specifications, and let $m'$ be a $\varphi'$-model such that $m' \models M'$. Then $m'\restriction_h \models M$.

**Proof:** It follows from the following:

$$
\begin{array}{llll}
m' \models (\varphi', \Sigma', A') & \text{iff} & m' \models_{\varphi'} Th^{\Sigma'}_{\varphi'}(A') & \text{(Definition 44)} \\
 & \text{implies} & m' \models_{\varphi'} h(Th^{\Sigma}_{\varphi}(A)) & (h \text{ is a morphism in } \mathbf{MSpec}) \\
 & \text{iff} & m'\restriction_h \models_{\varphi} Th^{\Sigma}_{\varphi}(A) & \text{(Satisfaction Condition)} \\
 & \text{iff} & m'\restriction_h \models (\varphi, \Sigma, A) & \text{(Definition 44)}
\end{array}
$$

□

The proposition above proves that, for any institution with inclusions, the model functor **Mod** extends to **MSpec**, by mapping a module specification $M$ to the full subcategory $\mathbf{Mod}(M)$ of $\mathbf{Mod}(\varphi)$ formed by the $\varphi$-models that satisfy $M$.

Now we are ready to introduce a very important concept, the conservatism of a module specification, which plays a central role in the basic module operations. But first, let us discuss about syntactic conservatism, a notion that appears often in the literature:

**Definition 46** A theory extension $(\Sigma, A) \hookrightarrow (\Sigma', A')$ is **syntactically conservative** iff $A = A' \cap \mathbf{Sen}(\Sigma)$. □

**Fact 47** If $M = (\varphi, \Sigma, A)$ is a module specification then $(\varphi, Vth(M)) \hookrightarrow (\Sigma, Th(M))$ is syntactically conservative.

**Proof:** This is because

$$
\begin{array}{lll}
Vth(M) & = & Th^{\Sigma}_{\varphi}(A) \\
 & = & \{a \in \mathbf{Sen}(\varphi) \mid A \models_{\Sigma} a\} \\
 & = & \{a \in \mathbf{Sen}(\varphi) \mid a \in Th(M)\} \\
 & = & Th(M) \cap \mathbf{Sen}(\varphi).
\end{array}
$$

□

As it is argued in [6], the syntactic conservatism is a necessary but insufficient condition for true conservatism. Therefore, it is not surprising that we need a stronger form of conservatism for module specifications:

**Definition 48** A module specification $M = (\varphi, \Sigma, A)$ is **conservative** if and only if the inclusion of theories $(\varphi, Vth(M)) \hookrightarrow (\Sigma, Th(M))$ is conservative (see Definition 23). □

It can be easily seen that a module which has only public features (or, in other words, a standard specification) is conservative.

# 4 Operations on Module Specifications

We formulate the usual operations on modules, which often appear in the literature, in our modularization approach. Two special points are emphasized: one of them regards the visible theorems of the composed module in terms of the visible theorems of the modules involved in the operation, and the other one regards the conservatism of the composed module (see the Propositions 54, 59, 64, 71 and 77).

Many of the results exposed in this section require some modules to be conservative; the most important one, Theorem 78, says that the instantiation of a parameterized module is a pushout in **MSpec**, as long as the body of the parameterized module and the interpretation module of the interface of the parameterized module are conservative. This justifies our pervasive care of conservatism.

## 4.1 Renaming

In the standard algebraic specifications, the renaming operation is very natural: all what a renaming by a morphism $h\colon \Sigma \to \Sigma'$ does to a specification $(\Sigma, A)$, is to rename each $\Sigma$-expression in $A$ accordingly, the result being $(\Sigma, A) * h = (\Sigma', h(A))$. In our framework, the renaming operation is more complicated and it is strictly dependent on the pushouts in **Sign**.

**Assumption:** Within this subsection, suppose that the working institution is semiexact and that **Sign** has pushouts which preserve inclusions.

**Definition 49** Let $M = (\varphi, \Sigma, A)$ be a module specification and $h\colon \varphi \to \varphi'$ be a morphism in **Sign**. The **renaming** of $M$ by $h$, written $M * h$, is the module specification $(\varphi', \Sigma_h, h_\Sigma(A))$ (see Notation 17). □

The operation above is well defined because $\varphi' \hookrightarrow \Sigma_h$ and $h_\Sigma(A)$ is a set of $\Sigma_h$-sentences. The morphism $h$ which is able to rename the visible signature $\varphi$ of $M$ to $\varphi'$ is first extended to the morphism $h_\Sigma$ on the whole working signature, and then $A$ is renamed by $h_\Sigma$.

**Proposition 50** In the context above, $h\colon M \to M * h$ is a morphism of module specifications.

**Proof:** It follows from the generalized closure lemma, which says that $h(Th^\Sigma_\varphi(A)) \subseteq Th^{\Sigma_h}_{\varphi'}(h_\Sigma(A))$. □

**Corollary 51** For any $\varphi_h$-model $m$, if $m \models M * h$ then $m\restriction_h \models M$.

**Proof:** It follows from Proposition 45. □

It is known and easy to prove that $(\Sigma, A) * h * g = (\Sigma, A) * (h; g)$ for standard specifications. This is not necessary true for module specifications, because the pushouts preserving the inclusions can be chosen in many different ways in **Sign**. But,

**Proposition 52** If the pushouts of inclusions in **Sign** are chosen in such a way that they can be composed vertically (see section 2.2), then $M * h * g = M * (h; g)$ where $M = (\varphi, \Sigma, A)$ is a module specification and $h\colon \varphi \to \varphi'$ and $g\colon \varphi' \to \varphi''$ are two signature morphisms.

**Proof:** It follows from the following:

$$\begin{array}{rcll} (\varphi, \Sigma, A) * h * g & = & (\varphi', \Sigma_h, h_\Sigma(A)) * g & \text{(Defintion 49)} \\ & = & (\varphi'', (\Sigma_h)_g, g_{(\Sigma_h)}(h_\Sigma(A))) & \text{(Defintion 49)} \\ & = & (\varphi'', \Sigma_{h;g}, (h;g)_\Sigma(A)) & \text{(by hypothesis)} \\ & = & (\varphi, \Sigma, A) * (h; g) & \text{(Defintion 49).} \end{array}$$

□

In the standard theory of institutions [7], the equality $h(A^\bullet)^\bullet = h(A)^\bullet$ holds for each specification $(\Sigma, A)$ and each signature morphism $h$ of source $\Sigma$. It can also be formulated for module specifications:

**Lemma 53** If $(\varphi, \Sigma, A)$ is a conservative module specification and $h: \varphi \to \varphi'$ is a signature morphism, then

$$Th_{\varphi'}^{\varphi'}(h(Th_\varphi^\Sigma(A))) = Th_{\varphi'}^{\Sigma_h}(h_\Sigma(A)).$$

**Proof:** From the generalized closure lemma (Lemma 33), $h(Th_\varphi^\Sigma(A))) \subseteq Th_{\varphi'}^{\Sigma_h}(h_\Sigma(A))$. Applying $Th_{\varphi'}^{\varphi'}$ to the inclusion above, we obtain $Th_{\varphi'}^{\varphi'}(h(Th_\varphi^\Sigma(A))) \subseteq Th_{\varphi'}^{\Sigma_h}(h_\Sigma(A))$ (because of Fact 32).

Conversely, let $a \in \mathbf{Sen}(\varphi')$ such that $h_\Sigma(A) \models_{\Sigma_h} a$ and let $m'$ be a $\varphi'$-model such that $m' \models_{\varphi'} h(Th_\varphi^\Sigma(A))$. Our goal is to show that $m' \models_{\varphi'} a$. From Satisfaction Condition we get $m'\restriction_h \models_\varphi Th_\varphi^\Sigma(A)$. Since $(\varphi, \Sigma, A)$ is conservative, it follows that there are some $\Sigma$-models $m$, such that $m\restriction_\varphi = m'\restriction_h$ and $m \models_\Sigma Th_\Sigma^\Sigma(A)$. Since the working institution is semiexact and considering how pullbacks behave in **Cat** (see Proposition 2), there is a $\Sigma_h$-model, let us say $m_h$, such that $m_h\restriction_{h_\Sigma} = m$ and $m_h\restriction_{\varphi'} = m'$. 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 $m_h \models_{\Sigma_h} h_\Sigma(A)$. Furthermore, $m_h \models_{\Sigma_h} a$ because $h_\Sigma(A) \models_{\Sigma_h} a$. Finally, $m_h \models_{\Sigma_h} \imath'(a)$, and so $m_h\restriction_{\imath'} \models_{\varphi'} a$; therefore $m' \models_{\varphi'} a$. □

The following proposition says that, under some hypotheses, the visible consequences (theorems) of a renamed module are given by the closure (over the target visible signature) of the image (by the renaming morphism) of the visible theorems of the initial module; also, it says that conservatism is preserved under renaming:

**Proposition 54** If $M = (\varphi, \Sigma, A)$ is a conservative module specification, then

1. $Vth(M * h) = h(Vth(M))^\bullet$ where the closure in the right-hand side is done over $\varphi'$-sentences.
2. $M * h$ is conservative.

**Proof:** The first assertion is a direct consequence of the Lemma 53. For the second one, let $m$ be a $\varphi'$-model of $Th_{\varphi'}^{\Sigma_h}(h_\Sigma(A))$, that is $m \models_{\varphi'} (\varphi', Vth(M * h))$. This is also equivalent to saying that $m$ is a model of $M * h$. By Corollary 51, $m\restriction_h$ is a $\varphi$-model of $Th_\varphi^\Sigma(A)$. Then by the conservatism of $(\varphi, \Sigma, A)$, there is a $\Sigma$-model $m_\Sigma$ of $A$ such that $m_\Sigma\restriction_\varphi = m\restriction_h$. But the pair of morphisms $h_\Sigma$ and $\varphi' \hookrightarrow \Sigma_h$ is a pushout of $h$ and $\varphi \hookrightarrow \Sigma$; therefore, by semiexactness and the construction of pullbacks in **Cat**, there is a $\Sigma_h$-model $m'$ such that $m'\restriction_{(h_\Sigma)} = m_\Sigma$ and $m'\restriction_{\varphi'} = m$. Then by the Satisfaction Condition, $m' \models_{\Sigma_h} h_\Sigma(A)$, that is $m'$ is a $\Sigma_h$-model of $Th_{\Sigma_h}^{\Sigma_h}(h_\Sigma(A))$. Therefore, for a $\varphi'$-model $m$ of $Vth(M * h)$ we found a $\Sigma_h$-model $m'$ of $Th(M * h)$ such that $m'\restriction_{\varphi'} = m$; this certifies that $M * h$ is conservative. □

**Corollary 55** If $M = (\varphi, \Sigma, A)$ is a conservative module specification, then $m \models M * h$ iff $m\restriction_h \models M$ for any $\varphi_h$-model $m$.

**Proof:** It follows from the following chain of equivalences:

| | | |
|---|---|---|
| $m\restriction_h \models M$ | iff | |
| $m\restriction_h \models_\varphi Vth(M)$ | iff | (Satisfaction Condition) |
| $m \models_{\varphi_h} h(Vth(M))$ | iff | |
| $m \models_{\varphi_h} h(Vth(M))^\bullet$ | iff | (Proposition 54) |
| $m \models_{\varphi_h} Vth(M * h)$ | iff | |
| $m \models M * h$ | | |

Observe that one implication is exactly the Corollary 51, and it does not need the conservatism of $M$. □

## 4.2 Hiding

The information hiding, a very important technique in modern computing, can be very naturally handled by our module specifications. To hide some features (sorts and operations) of a module, is the same thing as to allow only a reduced signature to be visible:

**Definition 56** If $M = (\varphi, \Sigma, A)$ is a module specification and $\psi$ is a subsignature of $\varphi$, then define $\psi \Box M$ as being the module specification $(\psi, \Sigma, A)$. $\Box$

Bergstra, Heering and Klop [1] called $\Box$ the "export operator", and Diaconescu, Goguen and Stefaneas [6] called it the "information hiding operator". We prefer the second variant for our approach.

**Proposition 57** If we let $\imath$ denote the inclusion $\psi \hookrightarrow \varphi$ in the definition above, then $\imath\colon \psi \Box M \to M$ is a morphism of module specifications.

**Proof:** The Fact 32 yields $Th^{\Sigma}_{\psi}(A) \subseteq Th^{\Sigma}_{\varphi}(A)$, that is $\imath$ is a morphism of module specifications. $\Box$

**Corollary 58** For any $\varphi$-model $m$, if $m \models M$ then $m{\restriction}_{\psi} \models \psi \Box M$.

**Proof:** It follows from Proposition 45. $\Box$

Of course, only a reduced number of visible theorems remain visible after an information hiding operation. The following proposition shows the relation between the visible theorems of $\psi \Box M$ and the visible theorems of $M$; also, it presents a sufficient condition under which $\psi \Box M$ is conservative:

**Proposition 59** Let $M = (\varphi, \Sigma, A)$ be a module specification, $\psi \hookrightarrow \varphi$ be an inclusion of signatures, and let $N$ be the module $(\psi, \varphi, Vth(M))$. Then:

1. $Vth(\psi \Box M) = Vth(N)$.
2. $\psi \Box M$ is conservative if $M$ and $N$ are conservative.

**Proof:** First, observe that $N = (\psi, \varphi, Vth(M))$ is a well-defined module specification because $\psi \hookrightarrow \varphi$ and $Vth(M) \subseteq \mathbf{Sen}(\varphi)$.

1. The equality is equivalent to $Th^{\Sigma}_{\psi}(A)) = Th^{\varphi}_{\psi}(Th^{\Sigma}_{\varphi}(A))$, which is true (see Fact 32).
2. Consider a $\psi$-model $m$ of $Th^{\Sigma}_{\psi}(A))$. Since $Th^{\Sigma}_{\psi}(A)) = Th^{\varphi}_{\psi}(Th^{\Sigma}_{\varphi}(A))$, then by the conservatism of $(\psi, \varphi, Th^{\Sigma}_{\varphi}(A))$, there are some $\varphi$-models $m'$ of $Th^{\varphi}_{\varphi}(Th^{\Sigma}_{\varphi}(A)) = Th^{\Sigma}_{\varphi}(A)$ such that $m'{\restriction}_{\psi} = m$. Then by the conservatism of $(\varphi, \Sigma, A)$ there are some $\Sigma$-models $m''$ of $Th^{\Sigma}_{\Sigma}(A)$ with $m''{\restriction}_{\varphi} = m'$. Therefore $m''{\restriction}_{\psi} = m$, that is $(\varphi, \Sigma, A)$ is conservative.

$\Box$

## 4.3 Enriching

Sometimes, we need to enrich a module specification by adding new operations and new sentences to the initial ones. The language LILEANA [22] implements this operation by adding a *partial* signature to the initial signature, obtaining a new signature, and then adding some sentences over the enriched signature to the initial ones. We do not offer support for partial signatures in our approach, but we can consider directly the extended signatures. A module specification can be enriched both with visible and with private features:

**Definition 60** If $M = (\varphi, \Sigma, A)$ is a module specification and $(\varphi', \Sigma', A')$ is another module specification such that $\varphi \hookrightarrow \varphi'$ and $\Sigma \hookrightarrow \Sigma'$, then define $M * (\texttt{add}\ \varphi', \Sigma', A')$ as being the module specification $(\varphi', \Sigma', A \cup A')$. $\Box$

It can be readily seen that the definition above is correct.

**Proposition 61** If $\imath$ is the inclusion $\varphi \hookrightarrow \varphi'$ above, then $\imath\colon M \to M * (\texttt{add}\ \varphi', \Sigma', A')$ is a morphism of module specifications.

**Proof:** We have to show that $Th^{\Sigma}_{\varphi}(A) \subseteq Th^{\Sigma'}_{\varphi'}(A \cup A')$; this follows from 2., 4. and 5. in Fact 32. $\Box$

**Corollary 62** For any $\varphi'$-model $m$, if $m \models M * (\texttt{add}\ \varphi', \Sigma', A')$ then $m{\restriction}_{\varphi} \models M$. $\Box$

Enriching is a technique which appears often both in computing and mathematics. In computing, it can be met mostly in *refinement*, when a formal specification is enriched with (eventually implementation) details. In mathematics, there are well-known situations in which a result cannot be proved as easily into only one theory (geometry, algebra, analysis, etc.) as into an enriched theory. For example, many geometry problems admit straightforward and neat proofs making use of trigonometry, complex numbers or algebraic tools (or combination of them), but they admit very difficult and awful proofs using only pure geometry. The visible segment of these enriched theories is still the pure geometry.

**Lemma 63** If $(\Sigma, \Sigma', A')$ is a conservative module specification and $A$ is a set of $\Sigma$-sentences, then

$$Th_{\Sigma}^{\Sigma'}(A \cup A') = Th_{\Sigma}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A')).$$

**Proof:** Since $A \subseteq Th\Sigma^{\Sigma'}(A \cup A')$ and $Th_{\Sigma}^{\Sigma'}(A') \subseteq Th_{\Sigma}^{\Sigma'}(A \cup A')$, by Fact 32 it follows that $Th_{\Sigma}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A')) \subseteq Th_{\Sigma}^{\Sigma'}(A \cup A')$. Conversely, let $a$ be a $\Sigma$-sentence in $Th_{\Sigma}^{\Sigma'}(A \cup A')$. In order to prove that $a$ is in $Th_{\Sigma}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A'))$, take a $\Sigma$-model $m$ of $A \cup Th_{\Sigma}^{\Sigma'}(A')$. Since $(\Sigma, \Sigma', A')$ is conservative, there is a $\Sigma'$-model $m'$ of $A'$ such that $m'\!\upharpoonright_{\Sigma}= m$. But $m \models_{\Sigma} A$, that is $m'\!\upharpoonright_{\Sigma} \models_{\Sigma} A$; 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, by the Satisfaction Condition we obtain that $m'\!\upharpoonright_{\Sigma} \models_{\Sigma} a$, that is $m\!\upharpoonright_{\Sigma} a$, which certifies that $a$ is in $Th_{\Sigma}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A'))$. □

A special case of enrichment arises often in practice, namely, the one in which a module needs to be enriched only with some private features; within our formalism (Definition 60), this means that $\varphi = \varphi'$. The following proposition shows a way to express the visible theorems of such an enriched module; also it presents a sufficient condition under which the enriched module is conservative:

**Proposition 64** Let $M = (\varphi, \Sigma, A)$ be a module specification which is going to be enriched with some new sentences $A'$ over an extension $\Sigma'$ of $\Sigma$, such that the module $M' = (\Sigma, \Sigma', A')$ is conservative. Then

1. $Vth(M * (\texttt{add}\ \varphi, \Sigma', A')) = Vth((\varphi, \Sigma, A \cup Vth(M')))$,
2. $M * (\texttt{add}\ \varphi, \Sigma', A')$ is conservative whenever $(\varphi, \Sigma, A \cup Vth(M'))$ is conservative.

**Proof:** Taking the $\varphi$-visible theorems of the two sides in the equality given by Lemma 63, we get:

$$Th_{\varphi}^{\Sigma'}(A \cup A') = Th_{\varphi}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A')).$$

1. It follows from the following equalities:

$$\begin{aligned} Vth(M * (\texttt{add}\ \varphi, \Sigma', A')) &= Vth((\varphi, \Sigma', A \cup A')) \\ &= Th_{\varphi}^{\Sigma'}(A \cup A') \\ &= Th_{\varphi}^{\Sigma}(A \cup Th_{\Sigma}^{\Sigma'}(A')) \\ &= Vth((\varphi, \Sigma, A \cup Vth(M'))). \end{aligned}$$

2. Let $m$ be a $\varphi$-model of $Vth(M * (\texttt{add}\ \varphi, \Sigma', A'))$. Then $m$ is also a $\varphi$-model of $Vth((\varphi, \Sigma, A \cup Vth(M')))$. Since $(\varphi, \Sigma, A \cup Vth(M'))$ is conservative, there is a $\Sigma$-model $m_{\Sigma}$ of $A \cup Vth(M')$ such that $m_{\Sigma}\!\upharpoonright_{\varphi}= m$. Now, because $M'$ is conservative, there is a $\Sigma'$-model $m'$ of $A'$ such that $m'\!\upharpoonright_{\Sigma}= m_{\Sigma}$. By the Satisfaction Condition, it follows that $m' \models_{\Sigma'} A$. Therefore, $m' \models_{\Sigma'} A \cup A'$ and, of course, $m'\!\upharpoonright_{\varphi}= m$.

□

The following two subsections are dedicated to two of the most important module operations, the aggregation and the parameterization. Because of their complexity, the working institution needs one more property; even if not all of the results need it, we make the following assumption in order to get a simpler presentation:

**Assumption:** From now on in the paper, suppose that the working institution is semiexact.

## 4.4 Aggregation

Module aggregation is the flat combination of modules. In practical examples, usually the modules have some common inherited modules; these inherited modules are shared in the aggregation (i.e., they appear only once), but all the symbols introduced in a module are tagged with the name of that module to avoid conflicts of symbol names in the aggregations. Because of this strategy, the operation of "putting together" two or more modules reduces to ordinary unions of signatures and sentences; only the symbols in the shared inherited modules overlap. This allows us to formalize this concept at institutional level:

**Definition 65** For any two module specifications $M = (\varphi, \Sigma, A)$ and $M' = (\varphi', \Sigma', A')$, we define their **aggregation** $M + M'$ as being the module specification $(\varphi \cup \varphi', \Sigma \cup \Sigma', A \cup A')$. $\square$

Notice that the definition above is correct because $\varphi \cup \varphi' \hookrightarrow \Sigma \cup \Sigma'$ by Fact 4, and $A \cup A'$ is a set of $\Sigma \cup \Sigma'$-sentences as **Sen** preserves inclusions.

**Fact 66** Aggregation is commutative, associative and idempotent. $\square$

Enriching is a special case of aggregation in our institutional approach, because $M * (\texttt{add}\ \varphi', \Sigma', A') = M + (\varphi', \Sigma', A')$. But this is only a syntactic issue, because the theory we develop for aggregation suppose that the modules involved do not have any common private symbols; this does not happen for enriching viewed as a special kind of aggregation.

**Proposition 67** If one lets $\imath$ and $\imath'$ denote the inclusions $\varphi \hookrightarrow \varphi \cup \varphi'$ and $\varphi' \hookrightarrow \varphi \cup \varphi'$ in the definition above, then $\imath \colon M \to M + M'$ and $\imath' \colon M' \to M + M'$ are morphisms of module specifications.

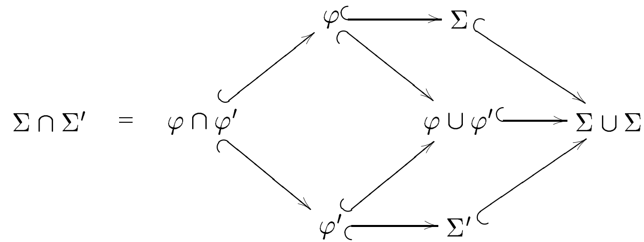
**Proof:** This is because $Th_{\varphi}^{\Sigma}(A) \subseteq Th_{\varphi \cup \varphi'}^{\Sigma \cup \Sigma'}(A \cup A')$ and $Th_{\varphi'}^{\Sigma'}(A') \subseteq Th_{\varphi \cup \varphi'}^{\Sigma \cup \Sigma'}(A \cup A')$ (see 2., 4. and 5. in Fact 32). $\square$

**Corollary 68** If $m \models M + M'$ then $m\restriction_{\varphi} \models M$ and $m\restriction_{\varphi'} \models M'$. $\square$

We call the following result "theorem" because of its crucial role in what follows. Informally, it says that if two conservative modules have no common private symbols, then each model (over the union of the visible signatures) of both sets of the visible theorems can be extended to a model (over the working signatures) of both sets of working theorems:

**Theorem 69** If $M = (\varphi, \Sigma, A)$ and $M' = (\varphi', \Sigma', A')$ are two conservative module specifications such that $\varphi \cap \varphi' = \Sigma \cap \Sigma'$, then for any $\varphi \cup \varphi'$-model $m$ of both $Vth(M)$ and $Vth(M')$ as sets of $\varphi \cup \varphi'$-sentences, there is a $\Sigma \cup \Sigma'$-model $m'$ of both $Th(M)$ and $Th(M')$ as sets of $\Sigma \cup \Sigma'$-sentences, such that $m'\restriction_{\varphi \cup \varphi'} = m$.

**Proof:** By the Satisfaction Condition, $m\restriction_{\varphi} \models_{\varphi} Th_{\varphi}^{\Sigma}(A)$ and $m\restriction_{\varphi'} \models_{\varphi'} Th_{\varphi'}^{\Sigma'}(A')$. Since $(\varphi, \Sigma, A)$ and $(\varphi', \Sigma', A')$ are conservative, there are some $\Sigma$-models $m_{\Sigma}$ of $A$ and some $\Sigma'$-models $m_{\Sigma'}$ of $A'$ such that $m_{\Sigma}\restriction_{\varphi} = m\restriction_{\varphi}$ and $m_{\Sigma'}\restriction_{\varphi'} = m\restriction_{\varphi'}$.




Then by the functoriality of the reducts,

$$
\begin{aligned}
m_{\Sigma}\restriction_{\varphi \cap \varphi'} &= (m_{\Sigma}\restriction_{\varphi})\restriction_{\varphi \cap \varphi'} \\
&= (m\restriction_{\varphi})\restriction_{\varphi \cap \varphi'} \\
&= m\restriction_{\varphi \cap \varphi'} \\
&= (m\restriction_{\varphi'})\restriction_{\varphi \cap \varphi'} \\
&= (m_{\Sigma'}\restriction_{\varphi'})\restriction_{\varphi \cap \varphi'} \\
&= m_{\Sigma'}\restriction_{\varphi \cap \varphi'}
\end{aligned}
$$

Since $\Sigma \cap \Sigma' = \varphi \cap \varphi'$, by strongness, semiexactness and 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 $m'\lceil_{\Sigma'} \models_{\Sigma'} A'$. Then by the Satisfaction Condition, $m' \models_{\Sigma \cup \Sigma'} A \cup A'$; it is left to the reader to observe that this is similar to saying that $m'$ is a model of both $Th(M)$ and $Th(M')$. It can be readily checked that $(m'\lceil_{\varphi \cup \varphi'})\lceil_{\varphi} = m\lceil_{\varphi}$ and $(m'\lceil_{\varphi \cup \varphi'})\lceil_{\varphi'} = m\lceil_{\varphi'}$; therefore $m'\lceil_{\varphi \cup \varphi'}$ verifies the conditions which are uniquely verified by $m$ (because the union of $\varphi$ and $\varphi'$ is a pushout of their intersection, and because of the strongness and semiexactness of the institution and the construction of pullbacks in **Cat**). Thus $m'\lceil_{\varphi \cup \varphi'} = m$. □

The relation $(A \cup A')^{\bullet} = (A^{\bullet} \cup A'^{\bullet})^{\bullet}$ holds for any standard specifications $(\Sigma, A)$ and $(\Sigma', A')$. It does not necessarily hold for module specifications. But,

**Lemma 70** Let $(\varphi, \Sigma, A)$ and $(\varphi', \Sigma', A')$ be two conservative module specifications such that $\varphi \cap \varphi' = \Sigma \cap \Sigma'$. Then

$$Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup A') = Th^{\varphi \cup \varphi'}_{\varphi \cup \varphi'}(Th^{\Sigma}_{\varphi}(A) \cup Th^{\Sigma'}_{\varphi'}(A')).$$

**Proof:** Obviously, $Th^{\Sigma}_{\varphi}(A) \subseteq Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup B)$ and also $Th^{\Sigma'}_{\varphi'}(B) \subseteq Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup B)$, so $Th^{\varphi \cup \varphi'}_{\varphi \cup \varphi'}(Th^{\Sigma}_{\varphi}(A) \cup Th^{\Sigma'}_{\varphi'}(B)) \subseteq Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup B)$.

Conversely, let us consider a $\varphi \cup \varphi'$-sentence $a$ such that $A \cup A' \models_{\Sigma \cup \Sigma'} a$, and let $m$ be a $\varphi \cup \varphi'$-model for $Th^{\Sigma}_{\varphi}(A)$ and $Th^{\Sigma'}_{\varphi'}(A')$. According to Theorem 69, there exists a $\Sigma \cup \Sigma'$-model $m'$ of $A \cup A'$ such that $m'\lceil_{\varphi \cup \varphi'} = m$. Then $m' \models_{\Sigma \cup \Sigma'} a$, and so by the Satisfaction Condition, $m'\lceil_{\varphi \cup \varphi'} \models_{\varphi \cup \varphi'} a$, that is $m \models_{\varphi \cup \varphi'} a$. Consequently, $a$ is in $Th^{\varphi \cup \varphi'}_{\varphi \cup \varphi'}(Th^{\Sigma}_{\varphi}(A) \cup Th^{\Sigma'}_{\varphi'}(A'))$ and it certifies that $Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup A') \subseteq Th^{\varphi \cup \varphi'}_{\varphi \cup \varphi'}(Th^{\Sigma}_{\varphi}(A) \cup Th^{\Sigma'}_{\varphi'}(A'))$. □

In terms of aggregations, we get:

**Proposition 71** Let $M = (\varphi, \Sigma, A)$ and $M' = (\varphi', \Sigma', A')$ be two conservative module specifications, such that $\varphi \cap \varphi' = \Sigma \cap \Sigma'$. Then

1. $Vth(M + M') = (Vth(M) \cup Vth(M'))^{\bullet}$, where the closure in the right-hand side is done over $\varphi \cup \varphi'$-sentences.
2. $M + M'$ is conservative.

**Proof:**

1. It follows immediately from Lemma 70.
2. Take a $\varphi \cup \varphi'$-model $m$ of $Th^{\Sigma \cup \Sigma'}_{\varphi \cup \varphi'}(A \cup A')$. Then $m$ is also a $\varphi \cup \varphi'$-model of $Th^{\Sigma}_{\varphi}(A)$ and $Th^{\Sigma'}_{\varphi'}(A')$, and by Theorem 69 there is a $\Sigma \cup \Sigma'$-model $m'$ of $A \cup A'$ such that $m'\lceil_{\varphi \cup \varphi'} = m$. Therefore, $m'$ is a $\Sigma \cup \Sigma'$-model of $Th^{\Sigma \cup \Sigma'}_{\Sigma \cup \Sigma'}(A \cup A')$; this certifies that $(\varphi, \Sigma, A) + (\varphi', \Sigma', A')$ is conservative.

□

**Corollary 72** In the context of the proposition above, if $m$ is a $\varphi \cup \varphi'$ model then $m \models M + M'$ iff $m\lceil_{\varphi} \models M$ and $m\lceil_{\varphi'} \models M'$

**Proof:** It follows from the following equivalences:

| | | |
|---|---|---|
| $m \models M + M'$ | iff | |
| $m \models_{\varphi \cup \varphi'} Vth(M + M')$ | iff | (Proposition 71) |
| $m \models_{\varphi \cup \varphi'} (Vth(M) \cup Vth(M'))^{\bullet}$ | iff | |
| $m \models_{\varphi \cup \varphi'} Vth(M) \cup Vth(M')$ | iff | |
| $m \models_{\varphi \cup \varphi'} Vth(M)$ and $m \models_{\varphi \cup \varphi'} Vth(M')$ | iff | |
| $m\lceil_{\varphi} \models_{\varphi} Vth(M)$ and $m\lceil_{\varphi'} \models_{\varphi'} Vth(M')$ | iff | |
| $m\lceil_{\varphi} \models M$ and $m\lceil_{\varphi'} \models Vth(M')$ | | |

Only one implication is interesting in this corollary, the other one being the Corollary 68 which needs neither conservatism nor $\varphi \cap \varphi' = \Sigma \cap \Sigma'$. □

**Corollary 73** If the working institution is distributive and $M_j = (\varphi_j, \Sigma_j, A_j)$ for $j \in 1..n$ are $n$ conservative module specifications such that $\Sigma_i \cap \Sigma_j = \varphi_i \cap \varphi_j$ for all $i, j \in 1..n$ with $M_i \neq M_j$, then:

1. $Vth(M_1 + \cdots + M_n) = (Vth(M_1) \cup ... \cup Vth(M_n))^\bullet$, where the closure is done over $\varphi_1 \cup ... \cup \varphi_n$-sentences,
2. $M_1 + \cdots + M_n$ is conservative, and
3. $m \models M_1 + \cdots + M_n$ iff $m\restriction_{\varphi_j} \models M_j$ for all $j \in 1..n$, where $m$ is a $\varphi_1 \cup ... \cup \varphi_n$-model.

□

## 4.5 Parameterization

One of the most effective ways to reuse software is parameterization.

**Assumption:** Within this subsection, suppose[9] that the category **Sign** in the working institution has pushouts which preserve inclusions, and an initial object $\emptyset$.

**Definition 74** A **parameterized module specification** $M[\alpha_1 :: P_1, ..., \alpha_n :: P_n]$ is a set of morphisms of module specifications $\alpha_j; \imath_j : P_j \to M$ where $M = (\varphi, \Sigma, A)$ and $P_j = (\pi_j, \Pi_j, B_j)$ for $j \in 1..n$, such that:

- $\alpha_j : \pi_j \to \varphi_j$ are isomorphisms of signatures,
- $\imath_j : \varphi_j \hookrightarrow \varphi$ are inclusions of signatures, and
- $\varphi_1, ..., \varphi_n$ are disjoint.

We say that $M$ is **parameterized** by $\alpha_1, ..., \alpha_n$. $P_1, ..., P_n$ are called **interfaces** and $M$ **body**. □

This definition implies automatically that $\alpha_j(Vth(P_j) \subseteq Vth(M)$ for all $j \in 1..n$.

**Definition 75** Given a parameterized module $M[\alpha_1 :: P_1, ..., \alpha_n :: P_n]$ as above and morphisms of module specifications $h_j : P_j \to M_j$ with $M_j = (\psi_j, \Omega_j, A_j)$ for all $j \in 1..n$, the **instantiation** of $M$ by $h_1, ..., h_n$, written $M[h_1, ..., h_n]$, is the module

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

where $h = \bigcup_{j=1}^n \alpha_j^{-1}; h_j$ (see Definition 9 and Proposition 12, and also Notation 17):

$$\begin{array}{ccccccc}
\varphi_i & \hookrightarrow & \bigcup_{j=1}^n \varphi_j & \hookrightarrow & \varphi & \hookrightarrow & \Sigma \\
\downarrow{\scriptstyle \alpha_i^{-1};h_i} & & \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle h_\varphi} & & \downarrow{\scriptstyle (h_\varphi)_\Sigma} \\
\psi_i & \hookrightarrow & \bigcup_{j=1}^n \psi_j & \hookrightarrow & \varphi_h & \hookrightarrow & \Sigma_{(h_\varphi)}
\end{array}$$

□

Notice that since $\varphi_1, ..., \varphi_n$ are disjoint, by 2. in Proposition 12 $h = \bigcup_{j=1}^n \alpha_j^{-1}; h_j$ exists and it is well defined. Therefore, the instantiation module $M[h_1, ..., h_n]$ is obtained renaming $M$ by $h_\varphi$ and then adding all modules $M_j$.

**Proposition 76** In the context of Definition 75,

1. $h_\varphi : M \to M[h_1, ..., h_n]$ is a morphism of module specifications.
2. $\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h$ is a morphism of module specifications, from $M_1 + \cdots + M_n$ to $M[h_1, ..., h_n]$.
3. $M[h_1, ..., h_n] = M * h_\varphi + M_1 + \cdots + M_n$.

**Proof:** 1. $h_\varphi : M \to M[h_1, ..., h_n]$ is a morphism of module specifications because:

$$\begin{array}{rcll}
h_\varphi(Vth(M)) & = & h_\varphi(Th_\varphi^\Sigma(A)) & \\
& \subseteq & Th_{\varphi_h}^{\Sigma_{(h_\varphi)}}((h_\varphi)_\Sigma(A)) & \text{(Lemma 33)} \\
& \subseteq & Th_{\varphi_h}^{\Sigma_{(h_\varphi)} \cup \bigcup_{j=1}^n \Omega_j}((h_\varphi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \text{(2. and 5. in Fact 32)} \\
& = & Vth(M[h_1, ..., h_n]) &
\end{array}$$

[9] In addition to semiexactness.

2. It is straightforward, since $Th^{\bigcup_{j=1}^n \Omega_j}_{\bigcup_{j=1}^n \psi_j}(\bigcup_{j=1}^n A_j) \subseteq Th^{\Sigma_{(h_\varphi)} \cup \bigcup_{j=1}^n \Omega_j}_{\varphi_h}((h_\varphi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j)$ by 2., 4. and 5. in Fact 32.

3. It follows from the following equalities:

$$
\begin{array}{lll}
M * h_\varphi + M_1 + \cdots + M_n = & & \\
= & (\varphi, \Sigma, A) * h_\varphi + (\psi_1, \Omega_1, A_1) + \cdots + (\psi_n, \Omega_n, A_n) & \\
= & (\varphi_h, \Sigma_{(h_\varphi)}, (h_\varphi)_\Sigma(A)) + (\psi_1, \Omega_1, A_1) + \cdots + (\psi_n, \Omega_n, A_n) & \text{(Definition 49)} \\
= & (\varphi_h \cup \bigcup_{j=1}^n \psi_j, \Sigma_{(h_\varphi)} \cup \bigcup_{j=1}^n \Omega_j, (h_\varphi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \text{(Definition 65)} \\
= & (\varphi_h, \Sigma_{(h_\varphi)} \cup \bigcup_{j=1}^n \Omega_j, (h_\varphi)_\Sigma(A) \cup \bigcup_{j=1}^n A_j) & \text{(2. in Fact 4)} \\
= & M[h_1, ..., h_n] & \text{(Definition 75)}
\end{array}
$$

□

The following proposition expresses the visible theorems of the instantiation module in terms of the visible theorems of the modules involved; it also shows that the instantiated module can be conservative even if the interface of the parameterized module is not conservative:

**Proposition 77** In the same context of Definition 75, if the working institution is distributive and

- $M, M_1, ..., M_n$ are conservative,
- $\Sigma_{(h_\varphi)} \cap \Omega_j = \psi_j$ for all $j \in 1..n$, and
- $\Omega_i \cap \Omega_j = \psi_i \cap \psi_j$ for all $i, j \in 1..n$, with $M_i \neq M_j$,

then

1. $Vth(M[h_1, ..., h_n]) = (h_\varphi(Vth(M)) \cup \bigcup_{j=1}^n Vth(M_j))^\bullet$, where the closure is done over $\varphi_h$-sentences, and
2. $M[h_1, ..., h_n]$ is conservative, and
3. $m \models M[h_1, ..., h_n]$ iff $m{\restriction}_{h_\varphi} \models M$ and $m{\restriction}_{\varphi_j} \models M_j$ for all $j \in 1..n$, where $m$ is a $\varphi_h$-model.

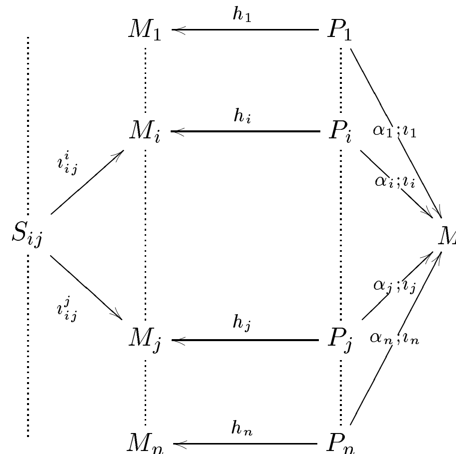
**Proof:** By Proposition 54, $Vth(M * h_\varphi) = h_\varphi(Vth(M))^\bullet$ and $M * h_\varphi$ is conservative, where the closure is done over $\varphi_h$-sentences. Since $M[h_1, ..., h_n] = M * h_\varphi + M_1 + \cdots + M_n$, iteratively applying Proposition 71 we get that $Vth(M[h_1, ..., h_n]) = (h_\varphi(Vth(M))^\bullet \cup \bigcup_{j=1}^n Vth(M_j))^\bullet$ and $M[h_1, ..., h_n]$ is conservative, where the closures are over $\varphi_h$-sentences. The rest follows from 2. in Fact 20.

3 follows immediately from the Corollaries 55 and 72. □

The hypothesis $\Sigma_{(h_\varphi)} \cap \Omega_j = \psi_j$ in the proposition above is not restrictive at all. It says that the private symbol names in $M$ should be eventually changed in the instantiated module, to avoid conflicts with private symbols in $M_j$.

A very important property of parameterization says that the instantiatiated module is a colimit:

**Theorem 78** In the context of Definition 75, if $S_{ij}$ are $\psi_i \cap \psi_j$-modules such that $Vth(S_{ij}) \subseteq Vth(M_i) \cap Vth(M_j)$ (we think of $S_{ij}$ as being the shared features of $M_i$ and $M_j$) for all $i, j \in 1..n$, then $M[h_1, ..., h_n]$ is a colimit of the diagram:

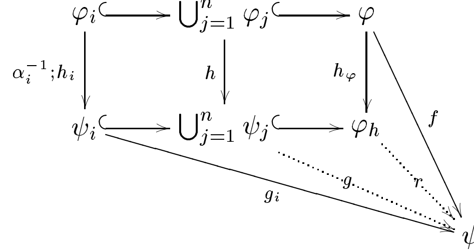


where $\imath^i_{ij}$ is the inclusion $\psi_i \cap \psi_j \hookrightarrow \psi_i$, for all $i, j \in 1..n$.

**Proof:** Notice that $\imath^i_{ij}\colon S_{ij} \to M_i$ are morphisms of modules. Also, notice that 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 all $i \in 1..n$, and
- $\imath^i_{ij}; g_i = \imath^j_{ij}; g_j$ for all $i, j \in 1..n$.

The following figure might help the reader follow the rest of the proof:



First, let us show that $h_\varphi\colon M \to M[h_1, ..., h_n]$ and $\psi_i \hookrightarrow \varphi_h\colon M_i \to M[h_1, ..., h_n]$ for $i \in 1..n$ is a cocone:

- Let $i \in 1..n$. Then,

$$\begin{array}{rcl} h_i; (\psi_i \hookrightarrow \varphi_h) & = & (\alpha_i; \alpha_i^{-1}); h_i; ((\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j); (\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h)) \\ & = & \alpha_i; ((\alpha_i^{-1}; h_i); (\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j)); (\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h) \\ & = & \alpha_i; ((\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); h); (\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h) \\ & = & \alpha_i; (\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); (h; (\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h)) \\ & = & \alpha_i; (\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); ((\bigcup_{j=1}^n \varphi_j \hookrightarrow \varphi); h_\varphi) \\ & = & \alpha_i; ((\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); (\bigcup_{j=1}^n \varphi_j \hookrightarrow \varphi)); h_\varphi \\ & = & \alpha_i; \imath_i; h_\varphi. \end{array}$$

- It is straightforward that $\imath^i_{ij}; (\psi_i \hookrightarrow \varphi_h) = \imath^j_{ij}; (\psi_j \hookrightarrow \varphi_h)$, because there is only one inclusion $\psi_i \cap \psi_j \hookrightarrow \varphi_h$.

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

$$\psi_i \hookleftarrow \psi_i \cap \psi_j \hookrightarrow \psi_j$$

for all $i, j \in 1..n$, so by 2 in Fact 8 there is a unique morphism of signatures, 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

$$\begin{array}{rcl} (\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); ((\bigcup_{j=1}^n \varphi_j \hookrightarrow \varphi); f) & = & ((\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); (\bigcup_{j=1}^n \varphi_j \hookrightarrow \varphi)); f \\ & = & \imath_i; f \\ & = & (\alpha_i^{-1}; \alpha_i); \imath_i; f \\ & = & \alpha_i^{-1}; (\alpha_i; \imath_i; f) \\ & = & \alpha_i^{-1}; (h_i; g_i) \\ & = & (\alpha_i^{-1}; h_i); g_i \\ & = & (\alpha_i^{-1};; h_i); ((\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j); g) \\ & = & ((\alpha_i^{-1};; h_i); (\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j)); g \\ & = & ((\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); h); g \\ & = & (\varphi_i \hookrightarrow \bigcup_{j=1}^n \varphi_j); (h; g), \end{array}$$

by 1 in Fact 8, one gets $(\bigcup_{j=1}^n \varphi_j \hookrightarrow \varphi); f = h; g$. But the rightmost square in the figure at the beginning of the proof is a pushout, so there is a unique $r\colon \varphi_h \to \psi$ such that $h_\varphi; r = f$ and $(\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h); r = g$.

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

$$\begin{array}{rcll} r(Vth(M[h_1, ..., h_n])) & = & r((h_\varphi(Vth(M)) \cup \bigcup_{j=1}^n Vth(M_j))^\bullet) & \text{(1 in Proposition 77)} \\ & \subseteq & r(h_\varphi(Vth(M)) \cup \bigcup_{j=1}^n Vth(M_j))^\bullet & \text{(Closure Lemma)} \\ & = & (r(h_\varphi(Vth(M))) \cup \bigcup_{j=1}^n r(Vth(M_j)))^\bullet & \\ & = & (f(Vth(M)) \cup \bigcup_{j=1}^n g_j(Vth(M_j)))^\bullet & \\ & \subseteq & Vth(C)^\bullet & \\ & = & Vth(C). & \end{array}$$

The uniqueness of $r\colon M[h_1, ..., h_n] \to C$ follows from the uniqueness of $r\colon \varphi_h \to \varphi$ as a morphism of signatures. Indeed, let $r'\colon M[h_1, ..., h_n] \to C$ be another morphism such that $h_\varphi; r' = f$ and $(\psi_i \hookrightarrow \varphi_h); r' = g_i$ for all $i \in 1..n$. Since the inclusions $\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j$ are an epimorphic family and

$$\begin{array}{rcl} (\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j); ((\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h); r') & = & ((\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j); (\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h)); r' \\ & = & (\psi_i \hookrightarrow \varphi_h); r' \\ & = & g_i \\ & = & (\psi_i \hookrightarrow \bigcup_{j=1}^n \psi_j); g, \end{array}$$

by 1 in Fact 8, $(\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h); r' = g$. Because of the uniqueness of $r\colon \varphi_h \to \psi$ with $h_\varphi; r = f$ and $(\bigcup_{j=1}^n \psi_j \hookrightarrow \varphi_h); r = g$, it follows that $r' = r$. □

In practical examples, very often modules are parameterized by only one parameter. If this is the case, then sharing between actual parameters is not a problem anymore, so a simpler result can be obtained:

**Corollary 79** If $M[\alpha_1 :: P_1]$ is a parameterized module and if $h_1\colon P_1 \to M_1$ is a morphism of module specifications, then the square:

$$\begin{array}{ccc} P_1 & \xrightarrow{\alpha_1; \imath_1} & M \\ {\scriptstyle h_1}\downarrow & & \downarrow{\scriptstyle h_\varphi} \\ M_1 & \xrightarrow[\imath]{} & M[h_1] \end{array}$$

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

**Proof:** It follows from Theorem 78, taking $S_{11} = M_1$. □

# 5 Conclusions and Future Research

We introduced the module specifications as a generalization of the standard specifications, having both public and private features, and then we showed that their category is equivalent to the category of theories. Basic module composition operations were investigated, showing that under an intrinsic module condition called conservatism, many properties of standard specifications can be lifted to module specifications. Our aim was to obtain general results related to module composition; as far as we know, most special cases of logic paradigms for programming languages verify the hypotheses of the results presented here.

One interesting area of further research is to explore how the results in the present paper can be extended to a multi-institutional framework (e.g., see [5, 21]), as an abstract formalism for multi-paradigm specification languages. Distributivity laws and refinement are other interesting areas open to further investigation in our approach.

Another direction[10] could be to associate to any institution with inclusions and pushouts which preserve inclusions a special institution where the category of signatures has inclusions of signatures as objects and pairs of morphisms given by the pushouts which preserves inclusions as morphisms. This could allow us to make use of the whole theory developed for standard specifications.

[10] Thanks to Răzvan Diaconescu for suggesting that.

# 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. Characterization of computable data types by means of a finite equational specification method. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, Seventh Colloquium*, pages 76–90. Springer, 1980. Lecture Notes in Computer Science, Volume 81.

[3] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems; part 2. Submitted for publication.

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

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

[6] 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.

[7] 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.

[8] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, to appear. Also UCSD Dept. Computer Science & Eng. Technical Report CS97–538, May 1997.

[9] Joseph Goguen and Grigore Roşu. Hiding more of hidden algebra. In *FM'99 – Formal Methods*, pages 1704–1719. Springer, 1999. Lecture Notes in Computer Sciences, Volume 1709, Proceedings of World Congress on Formal Methods, Toulouse, France.

[10] Joseph Goguen and Will Tracz. An implementation-oriented semantics for module composition, 1997. In preparation.

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

[12] Hendrik Hilberdink. Inclusion systems, 1996. Unpublished paper.

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

[14] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium 1987*, pages 275–329. North-Holland, 1989.

[15] 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.

[16] István Németi. On notions of factorization systems and their applications to cone-injective subcategories. *Periodica Mathematica Hungarica*, 13(3):229–335, 1982.

[17] Grigore Roşu. Axiomatizability in inclusive equational logic, 1996. Submitted to publication.

[18] Grigore Roşu and Joseph Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, pages 252–267. Springer, 2000. Lecture Notes in Artificial Intelligence, Volume 1761; papers from a conference held in Vienna, November 1998.

[19] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Control*, 76:165–210, 1988.

[20] Andrzej Tarlecki. Bits and pieces of the theory of institutions. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Proceedings, Summer Workshop on Category Theory and Computer Programming*, pages 334–360. Springer, 1986. Lecture Notes in Computer Science, Volume 240.

[21] 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. Proceedings of 11th Workshop on Specification of Abstract Data Types. Oslo, Norway, September 1995.

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

# Contents

**1 Introduction** **1**

**2 Preliminaries** **2**
2.1 Category Theory . . . . . 2
2.2 Inclusions . . . . . 2
2.3 Institutions . . . . . 6

**3 Module Specifications** **8**
3.1 Visible Theorems . . . . . 8
3.2 Motivation . . . . . 11
3.3 Definition and Properties of Module Specifications . . . . . 12

**4 Operations on Module Specifications** **14**
4.1 Renaming . . . . . 14
4.2 Hiding . . . . . 15
4.3 Enriching . . . . . 16
4.4 Aggregation . . . . . 18
4.5 Parameterization . . . . . 20

**5 Conclusions and Future Research** **23**