Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 304 (2014) 127–149

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Abstract Semantics for $\mathbb{K}$ Module Composition

Codruta Girlea[1] Grigore Rosu[2]

Computer Science Department
University of Illinois at Urbana - Champaign
Urbana, USA

Abstract

A structured $\mathbb{K}$ definition is easier to write, understand and debug than one single module containing the whole definition. Furthermore, modularization makes it easy to reuse work between definitions that share some principles or features. Therefore, it is useful to have a semantics for module composition operations that allows the properties of the resulting modules to be well understood at every step of the composition process. This paper presents an abstract semantics for a module system proposal for the $\mathbb{K}$ framework. It describes $\mathbb{K}$ modules and module transformations in terms of *institution*-based model theory introduced by Goguen and Burstall.

Keywords: Module System, $\mathbb{K}$, Semantics, Institutions, Abstract Modules

1 Introduction

The $\mathbb{K}$ executable semantic framework [1] allows one to define programming languages, calculi and even algorithms, by means of configurations, computations and rules. For example, in order to define a certain programming language, one must: define the syntax of the language; define the initial configuration of any program written in that language — thus also establishing the structure of its configurations; and specify the semantics of programs by means of rewrite rules that show how the language constructs affect the program configuration. More details on the $\mathbb{K}$ framework can be found in [1].

The $\mathbb{K}$ technique allows for modularization and parallelism by means of *configuration abstraction*, which means that in order to add features to a certain programming language, one only has to add the desired rules and configuration cells, without changing anything that has been written so far. But adding a certain feature still

[1] Email: girlea2@illinois.edu
[2] Email: grosu@illinois.edu

http://dx.doi.org/10.1016/j.entcs.2014.05.007

involves editing the existing definition, which means creating different extensions of a language, with different sets of features; furthermore, if one wants to add, for example, barriers to two or more programming languages that support threads, one would still have to do it separately for each of those languages.

Ideally, in order to add new features to programming languages, one should not have to write anything in the existing definitions. This can be done by writing a module for each feature, and in order to use the extended language, by using the existing definition together with the modules that define the features — as will be seen, this operation is called *aggregation*. To ensure that the semantics is preserved between the base definition and the feature definitions, each module has a section that specifies which constructs must be available to it and what behavior it expects from them — this is the 'required' part of the module. To ensure proper encapsulation, every module should state which constructs and behaviors it defines, that are accessible to any other module — this is the 'exported' part of the module.

For example, in order to add the '`++`' increment operation to a very simple imperative language, IMP (see, e.g., [1]), one would define a module that:

- requires sorts `Int`, `AExp`, `Id`, `Map`; an operation `+` on `Int`; two configuration cells `<env/>` and `<store/>`, each holding a `Map`; additionally, certain properties of these operations may also be required;
- adds `++Id` as a new `AExp` language construct and defines its desired behavior,

```
rule <k> ++X:Id => I + 1 ...</k>
     <env>... X |-> N:Int ...</env>
     <store>... N |-> (I:Int => I + 1) ...</store>
```

- exports `AExp` and `++`.

Barriers are synchronization mechanisms that ensure that all threads have reached a certain point in a program before the computation in any of the threads continues. In order to add barrier as a language construct to IMP, one would only require a sort `Stmt` and a cell `<thread/>`, define the behavior of a new language construct `barrier` and export `barrier` and `Stmt`.

As mentioned before, in order to get the extended language, one only has to aggregate the feature modules and the module of the basic definition; the names of the required definition in the feature modules need not be the same as the names in the original definition — before the aggregation mentioned above, a renaming step can be performed so that the proper names are given to the syntactic entities and cells. Furthermore, in some cases, the same feature modules can be reused, with different renamings, for different base languages.

Another way to extend a definition is to use enriching, directly — this is preferable when the features added are either not complex enough or have limited potential for reusability.

Another way to reuse a module is by hiding some features. For example, after defining exceptions in a functional programming language, by means of call-with-current-continuation (`call/cc`), one may want to hide the `call/cc` construct and allow users of the language to only use the exceptions provided.

The motivation for this paper comes from the need for a module system for $\mathbb{K}$, and the inspiration for our particular notation and format for modules comes from previous work by Hills and Rosu [9], where they describe the intended syntax and usage of $\mathbb{K}$ modules from a pragmatic point of view. Indeed, the distinction between the visible part and the rest of the signature reduces the interactions between modules, whereas allowing a required part simplifies dealing with those interactions, however complex, since one does not need to be concerned with the order in which modules are implemented. We analyze the theoretical properties of our module system using institution-based model theory [6]. The $\mathbb{K}$ Institution is assumed to have the desired properties — proving that this is indeed the case is left as future work, and so it the implementation of our module system. Given the nature of this work, although the $\mathbb{K}$ module system is the main motivation, we believe that the proposed abstract module system may prove useful to other systems as well.

Our semantics is similar to the module semantics described by Goguen and Rosu [7,3]. Thus, a module is seen as a presentation in a given institution (the definition of the module), where the visible part of the module (*visible signature* or *interface*) is a sub-signature of the module definition signature (the *working signature*), and the *visible theorems* are the restriction to this signature of the set of theorems of the module [7]. In addition to those, a $\mathbb{K}$ module may assume a part of its definition as already implemented and state this part as a *required* presentation. As such, the definition of a $\mathbb{K}$ module has the following general form:

```
module M {
    requires ρ, Kρ
    exports ψ
    Σ,K
    }
```

Σ, ψ and ρ are the *working*, *visible* and *required* signatures, respectively (where ψ and ρ are subsignatures of Σ) and $K\rho$, K are the *required* and *working* theorems, respectively.

We assume the institution we are working in has an inclusion system [2] on signatures, wherein each signature morphism σ can be factored (uniquely up to isomorphism) as the composition $\sigma = e; \iota$ of an abstract surjection e and an abstract inclusion ι. Furthermore, we assume the institution is *inclusive* [3] and that the model functor of the institution (Mod) preserves pushouts and coproducts.

The module operations we define the semantics of are: renaming, hiding, enriching and aggregation.

Renaming allows the reuse of modules with different names for the required and visible symbols and it only makes sense if it translates the symbols that those two signatures share in a consistent manner. Intuitively, renaming does not add new symbols to the signature, thus the morphims that define the renaming are surjections.

Hiding allows a part of the visible signature to no longer be visible in the new module.

Enriching, as opposed to hiding, adds new symbols and sentences to the module. One can also add new symbols and particularly sentences to the set of requirements, which is still, by definition, an enriching, but in this case the effect is that of constraining (if new sentences are added), as the set of elements required to define the module grows.

Aggregation allows two modules to be combined into one single module.

If a module defines everything it requires, it is called *complete*. Using all the above operations on a set of modules, one can define a structured module. One can obtain a complete structured module even if some or all the base modules used are incomplete.

1.1 Related Work

Probably the first module system for specification languages was described by Bergstra et. al. [11]. Their system is not based on institutions, but their module operations were used in a similar manner in most institution-based module systems to follow. They do not describe parametrization and, while not directly specifying a visible sub-signature, parts of the signature can be hidden using the hiding operation.

The first abstract module system using institution-based model theory was proposed by Diaconescu et. al. [2]; here, a module was just a presentation, consisting of a signature in a certain institution, and a set of sentences of the signature, specifying the behaviour of the constructs. Operations for sum, renaming and hiding are defined on modules and some properties of those operations such as distributive laws are analyzed.

The system proposed by Goguen et. al. [8] allows a module to specify its imported modules, as well as its visible part. The modules are defined here using partial signatures. The notion of an implementation module, for a specification module, is introduced. Hiding, enriching, renaming and aggregation operations are defined for specification modules, and renaming and aggregation of implementation modules are also introduced.

The module system of the CASL specification language [13] allows hiding and revealing sorts from a specification. These operations are supported by our module system as hiding and a particular type of enriching. CASL also support the *free* operation, that changes the semantics of a module to the initial/free model.

Maude's module system [12] allows module imports, as well as more general morphisms, parametrization and freeness constraints. Our system does not allow freeness constraints, or any operations that specifically target the semantics of a module. This kind of constraints may be worth exploring once the $\mathbb{K}$ institution has been defined and its liberality explored.

This work in this paper particular is an extension of the abstract module system introduced by Rosu and Goguen [7] [3] — in particular, the exported part of the module is defined in the same way as the visible signature, as a sub-signature of the working signature. In addition, we add the required part of the module, also

a sub-signature of the working signature, and the required behaviours, as a set of sentences. This addition makes it possible to assume that some elements a module relies on are already implemented and treat them as such, after stating the fact that they are required. Even though, at implementation time, these 'required' elements make that particular module incomplete, all the modules involved, implemented separately, will be aggregated into a final system that is complete and additionally makes it easier to handle the interdependecies between modules, particularly when these are too complex to structure using only imports and views.

Even though most operations used in the Rosu and Goguen's module system can be adapted without much difficulty to this new setting, keeping track of the required sub-signature leads to interesting situations and, in fact, to some particular cases of the module operations than can be significant operations in their own right. An example is enriching, where 'enriching' the required sub-signature and in particular the required behaviour has an opposite effect, of 'constraining' the elements needed for the module. Rosu and Goguen define the hiding, enriching, renaming, aggregation and parametrization operations for the modules they define. We only analyze hiding, enriching, renaming and aggregation here.

Parametrization is an interesting case because, again, there are two directions from which we can parametrize. We can parametrize a module in a way similar to the one described by Rosu and Goguen [7]. We need to make sure that when we instantiate the module, the module morphisms from parameters to the modules we use to instantiate are consistent on the required and visible part, in a manner similar to the way we will require renaming to be consistent. This way, the instantiated module will have a well defined required signature, including the (transformed) requirements of 'instantiated' parameters. On the other hand, we may be interested in parametrizing the required signature. We leave to future work an analysis of the interactions between requirements and parametrization, as well as the usefulness of including both or just one of the previously mentioned cases of parametrization.

ELAN [10] also uses a module system based on a visible part (specified by the keyword `global`) and a hidden signature (specified by the keyword `local`). The ELAN module system uses imports, but does not specify required elements for the module. Our module system allows visibility control by specifying the exported elements and by using the hiding and enriching operations. Furthermore, using a required signatures allows more flexibility in implementation.

2 Background

In this section we introduce some concepts needed for the work presented in this paper; we assume the reader familiar with basic concepts of category theory [14] and only go over the concepts to be used in the rest of the paper.

2.1 Inclusion systems

We use the inclusion systems as defined in [2] — intuitively, these systems generalize the 'natural' inclusion system of sets, where each function can be factorized as a

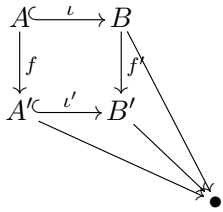

Fig. 1. Pushouts which preserve inclusions

composition of a surjection and an inclusion. Unions and intersections are also generalized as coproducts and respectively products of inclusions. Inclusive categories and functors are as defined in [3].

Definition 2.1 Let $\mathbb{C}$ be a category. An inclusion system is a pair of subcategories $(\mathcal{I}, \mathcal{E})$ of $\mathbb{C}$, called *inclusions* and respectively *surjections*, with the following properties:

- $|\mathbb{C}| = |\mathcal{I}| = |\mathcal{E}|$
- $\mathcal{I}(A, B)$ has at most one element for each $A, B \in |\mathbb{C}|$
- for every $A, B \in |\mathbb{C}|$, if both $\mathcal{I}(A, B)$ and $\mathcal{I}(B, A)$ are non-empty, then $A = B$
- for any morphism $f \in \mathbb{C}(A, B)$, there is a unique $C \in \mathbb{C}$ and a unique pair $(e, i) \in |\mathcal{E}(A, C)| \times |\mathcal{I}(C, B)|$ such that $e; i = f$
- $\mathcal{I}$ has finite coproducts

Definition 2.2 A category $\mathbb{C}$ is called *inclusive* (or it has *strong inclusions*) if there is a subcategory $\mathcal{I}$ of $\mathbb{C}$ such that the following hold:

- $\mathcal{I}$ is a poset
- $\mathcal{I}$ has finite products $\cap$ (called *intersections*)
- $\mathcal{I}$ has finite coproducts $\cap$ (called *unions*)
- for every pair $A, B \in |\mathbb{C}|$, $A \cup B$ is a pushout of $A \cap B$
- $\mathcal{I}$ has an initial object $\emptyset$

If $\mathcal{I}(A, B)$, then let $A \hookrightarrow B$ be the unique morphism in $\mathcal{I}(A, B)$.

Definition 2.3 Let $\mathbb{A}, \mathbb{B}$ be two categories. A functor $F : \mathbb{A} \to \mathbb{B}$ is called *inclusive* if it takes inclusions in $\mathbb{A}$ to inclusions in $\mathbb{B}$.

Definition 2.4 A category $\mathbb{C}$ has *pushouts which preserve inclusions* if:

- it is inclusive
- for any objects A, A', B in $|\mathbb{C}|$ and any morphisms $f : A \to A'$, $\iota : A \to B$, where ι is an inclusion, there is a pushout $\{B', f', \iota'\}$, with $f' : B \to B'$ and $\iota' : A' \to B'$, such that ι' is also an inclusion (see Figure 1)

2.2 Institutions

Institutions were introduced by Goguen and Burstall [6] as a means to represent logical systems in a unified, abstract manner: the syntax is given by a category of *signatures* and a functor that gives, for each signature, its set of *sentences*; the semantics is given by a functor that attaches a category of *models* to each signature, and by the *satisfaction* relation between models and sentences of the same signature. Many logics have been formalized as institutions, including equational logic [4], higher order logic [5] and many others [15].

Definition 2.5 An institution I is a tuple $(Sign, Sen, Mod, \models)$ where:

- $Sign$ is a category; the objects of $Sign$ are called *signatures*
- $Sen : Sign \to Set$ is a functor; the objects of $Sen(\Sigma)$, where $\Sigma \in |Sign|$ is a signature, are called the *sentences* of Σ
- $Mod : Sign^{op} \to \mathbb{C}at$ is a functor; the objects of $Mod(\Sigma)$, where $\Sigma \in |Sign|$ is a signature, are called the *models* of Σ
- for each $\Sigma \in Sign$, $\models$ is a relation $\models \subset |Mod(\Sigma)| \times |Sen(\Sigma)|$; if $M \in Mod(\Sigma)$, $\phi \in Sen(\Sigma)$ and $M \models \phi$, we say M satisfies ϕ.

For an institution $I = (Sign, Sen, Mod, \models)$, for $\Sigma, \Sigma' \in Sign$, $f : \Sigma \to \Sigma'$ and $M' \in |Mod(\Sigma')|$, $Mod(f)(M') \in Mod(\Sigma)$ is called the *reduct* of M' via f and is written $M' \restriction_f$. If the signature morphism is clear from the context, we will write $M' \restriction_\Sigma$. The intuition is that the reduct is that part of the model of the second signature, that can be recovered using only symbols from the first signature, and is usually 'smaller'. The meaning of this, as the actual definition of the reduct, varies across logics. A simple example is for the case of First Order Logic, for a signature morphimsm that adds a sort s to a signature Σ. Given a model M of $\Sigma \uplus \{s\}$, its reduct is a model that is exactly the same as M, but without the interpretation of s.

Conservativeness is a property of signature morphism that ensures that models are not 'lost' in translating between signatures. More specifically, it means that every model in the first signature has a corresponding model in the second, that reduces to it.

Definition 2.6 Given an institution $I = (Sign, Sen, Mod, \models)$, a signature morphism $f : \Sigma \to \Sigma'$ is *conservative* if for every model $M \in Mod(\Sigma)$, there is a model $M' \in Mod(\Sigma')$ such that $M' \restriction_f = M$

We further describe the notion of inclusive institution, which assumes an inclusion system on the category of signature, which translates in an intuitive manner to inclusions of respective sets of sentences and to interpretations.

Definition 2.7 An institution $I = (Sign, Sen, Mod, \models)$ is *inclusive* if:

- $Sign$ is an inclusive category;
- Sen is an inclusive functor;

$$\begin{array}{ccc} \Sigma, M_1 \restriction_f = M_2 \restriction_g & \xrightarrow{g} & \Sigma_2, M_2 \\ \Big\downarrow f & & \Big\downarrow g' \\ \Sigma_1, M_1 & \xrightarrow{f'} & \Sigma' \exists! M \end{array}$$

Fig. 2. Model amalgamation

- Mod preserves pushouts which preserve inclusions

Model amalgamation is a property that allows consistent aggregation of signatures - and eventually of specifications, as we will see. It essentially says that if two signatures 'share' some symbols, then any two models of the signatures that are consistent on the 'shared' symbols can be extended to the same model in the sum of the two signatures - and there is only one model in the sum of the signatures with this property.

Definition 2.8 An institution $I = (Sign, Sen, Mod, \models)$ has *model amalgamation* iff for every pushout square $(f : \Sigma \to \Sigma_1, g : \Sigma \to \Sigma_2, f' : \Sigma_1 \to \Sigma', g' : \Sigma_2 \to \Sigma')$ of signatures (see Figure 2), for every $M_1 \in |Mod(\Sigma_1)|$, $M_2 \in |Mod(\Sigma_2)|$ such that $M_1 \restriction_f = M_2 \restriction_g$, there is a unique model $M \in |Mod(\Sigma')|$ such that $M \restriction_{f'} = M_1$ and $M \restriction_{g'} = M_2$.

Without requiring uniqueness, this is called *weak* amalgamation. A more restrictive property similar to amalgamation is exactness:

Definition 2.9 An institution $I = (Sign, Sen, Mod, \models)$ is *exact* if the Mod functor preserves colimits, i.e. it translates signature colimits to limits in $\mathbb{C}at$.

A weaker form of exactness, called *semiexactness*, only requires Mod to preserve pushouts.

3 The Abstract Module System

In this section, we give the definition of an abstract module, show some of its properties and define the abstract module operations.

In the following we will assume we are working in an inclusive institution $I = (Sign, Sen, Mod, \models)$; let $\mathcal{I}$ be the inclusion subcategory and $\mathcal{E}$ the surjection subcategory. The inclusions will also be referred to as $\hookrightarrow$. In addition, we assume the category of signatures $Sign$ has strong inclusions and pushouts which preserve inclusions and that Mod preserves pushouts and coproducts.

Although we haven't yet formalized the $\mathbb{K}$ institution, we believe that it will satisfy many of these properties. The essential property here is the inclusive signature category, i.e. inclusions, unions and intersections of signatures, in the intuitive sense. We believe that this is a necessary and likely requirement for any module system, and that for $\mathbb{K}$, it will result from the inclusion system on sets. For this latter reason, the sentence functor is also expected to be inclusive, inclusions on signatures translating to inclusions of their sentence sets. It may be challenging to analyse the model functor's effect on pushouts and coproducts, but we do note

that preserving them is only necessary for aggregation. However, as also mentioned for model amalgamation, at least a weak version of this property is essential for aggregation in general.

3.1 The Modules

A module can be represented as:

```
module M {
    requires ρ, Kρ
    exports ψ
    Σ,K
    }
```

where Σ, ψ and ρ are the *working*, *visible* and *required* signatures, respectively (where ψ and ρ are subsignatures of Σ) and $K\rho$, K are the *required* and *working* theorems, respectively. Formally:

Definition 3.1 A module specification is a structure of the form $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ where:

- $\Sigma, \rho, \psi \in Sign$ and $\iota_\rho : \rho \hookrightarrow \Sigma$, $\iota_\psi : \psi \hookrightarrow \Sigma$ are signature inclusions; Σ is the *working signature* of the module, ρ is the *required signature* and ψ is the *visible signature* (or interface) of $\mathcal{M}$;
- $Th(\mathcal{M}) = K \in Sen(\Sigma)$ is the set of *working theorems* of the module ;
- $Rth(\mathcal{M}) = K_\rho \in Sen(\rho)$ is the set of assumed or *required theorems* of the module.

The *visible theorems* of $\mathcal{M}$ are therefore

$$Vth(\mathcal{M}) = K_\psi = \iota_\psi^{-1}((K \cup \iota_\rho(K_\rho))^\bullet) \tag{1}$$

where, for a set Γ of sentences with signature Σ, $\Gamma^\bullet$ is its semantic closure:

$$\Gamma^\bullet = \{\rho \in Sen(\Sigma) \mid \forall M \in Mod(\Sigma), \text{ if } M \models \Gamma \text{ then } M \models \rho\} \tag{2}$$

Below we prove that unions of semantically closed sets are also semantically closed, and in particular this holds for the visible theorems of a module.

Fact 3.2 *If $\Sigma \in Sign$ and $A, A' \in Sen(\Sigma)$ are closed sets of sentences, then:*

$$A \cap A' = (A \cap A')^\bullet$$

Proof.

The direct inclusion follows from the closure definition: $A \cap A' \subseteq (A \cap A')^\bullet$.

For the inverse inclusion, let $a \in A \cap A'$. Then $a \in A$ and $a \in A'$, and since A and A' are both closed, i.e. $A = A^\bullet$ and $A' = A'^\bullet$, this means

- $\forall \rho \in A, \forall m \in Mod(\Sigma)$ such that $m \models \rho$, it follows $m \models a$, and
- $\forall \rho' \in A', \forall m' \in Mod(\Sigma)$ such that $m' \models \rho'$, it follows $m' \models a$

This means that $\forall \rho \in A \cap A', \forall m \in Mod(\Sigma)$ such that $m \models \rho$, it follows $m \models a$, which means that $a \in (A \cap A')^\bullet$ □

Fact 3.3 *Given a module specification $\mathcal{M}$, its set of its visible theorems is semantically closed:*

$$Vth(\mathcal{M}) = Vth(\mathcal{M})^\bullet$$

Proof.

Let $\Gamma = K \cup \iota_\rho(K_\rho) \in Sen(\Sigma)$.

Since the working institution is inclusive, it follows

$$Vth(\mathcal{M}) = \iota_\psi^{-1}(\Gamma^\bullet) = Sen(\psi) \cap \Gamma^\bullet \tag{3}$$

Since $Sen(\psi)$ is trivially closed, using Fact 3.2 (note that K and $i_\rho(K_\rho)$ have the same signature Σ):

$$Vth(\mathcal{M})^\bullet = (Sen(\psi) \cap \Gamma^\bullet)^\bullet = Sen(\psi) \cap \Gamma^\bullet = Vth(\mathcal{M}) \tag{4}$$

□

A module is *complete* if it defines everything it requires. We need this notion because we are not working with partial signatures: we have defined the required signature to be a subsignature of the working signature; therefore, what we really describe when we write the definition is the part of the working signature not contained in the required signature, so if what the module defines does not in fact entail what it requires, the module is incomplete, and we cannot work with it. Ideally, after applying module operations, we will eventually obtain and work with complete modules.

Definition 3.4 We say a module $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ is *complete* if $\iota_\rho(K_\rho) \subseteq K^\bullet$

Consequently, the visible theorems of a complete module are $K_\psi = \iota_\psi^{-1}(K^\bullet)$

The module operations we will further define will be special cases of module morphisms, as defined below:

Definition 3.5 The module specification morphisms $(g, h) : (\rho, K_\rho, \psi, \Sigma, K) \to (\rho', K_{\rho'}, \psi', \Sigma', K')$ are pairs of morphisms where

- $g : (\rho, K_\rho) \to (\rho', K_{\rho'})$ is a presentation morphism
- $h : \psi \to \psi'$ is a signature morphism with $h(Vth(\mathcal{M})) \subseteq Vth(\mathcal{M}')$

So essentially, a module morphism specifies transformations on the visible, as well as on the required part of the module.

A model of a module must follow the behaviour specified by the visible theorems. Additionally, it must have a respective model that satisfies the required behaviour. This follows automatically if the model can be extended to the full, hidden signature.

Definition 3.6 Let $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ be a module specification and m a ψ model. Then model m satisfies the module specification, or $m \models \mathcal{M}$, if $m \models_\psi Vth(\mathcal{M})$ and there is an expansion m' of m to Σ such that $m' \restriction_\rho \models_\rho K_\rho$,

Note 1 *If all the inclusions are conservative signature morphisms, then* $m \models_\psi Vth(\mathcal{M})$ *is a sufficient condition for module satisfaction: if there is an expansion* m' *of* m *to* Σ*, then it follows using the definition of the satisfaction relation that* $m' \restriction_\rho \models_\rho K_\rho$.

3.2 The Module Operations

In this subsection, we will define and analyze the semantics of the following module operations: renaming, hiding, enriching and aggregation; parametrization will be analyzed as future work.

Renaming allows reusing modules by translating the names of the required and visible symbols. Intuitively, renaming does not add new symbols to the signature, thus the morphims that define the renaming are surjections. Hiding allows a part of the visible signature to be 'retracted' or hidden, thus no longer accessible to other modules. Enriching adds new constructs and behaviours to the module. Similarly to the way renaming can be seen as a 'surjection' on modules, hiding and enriching make use of inclusions. Aggregation allows two modules to be combined into one single module.

For the rest of this section, we will consider this small example:

```
module Xor
   requires sort Bool
   requires op or: Bool Bool -> Bool
   requires op not: Bool -> Bool
   requires op and: Bool Bool -> Bool
   exports op xor: Bool Bool -> Bool
   exports sort Bool
   rule X:Bool xor Y:Bool => (X and not Y) or (not X and Y)
end module
```

3.2.1 Renaming

Renaming only makes sense if it translates the symbols that those two signatures share in a consistent manner. Then, renaming can be performed on the union of the two signatures.

This is what the following lemma states:

Lemma 3.7 • *Let* $\rho, \psi \in Sign$ *and surjections* $g : \rho \to \rho'$, $h : \psi \to \psi'$ *such that if* $\iota_\rho; g = e_g; \iota_g$ *is the unique factorization of* $\iota_\rho; g$ *and* $\iota_\psi; h = e_h; \iota_h$ *is the unique factorization of* $\iota_\psi; h$*, then* $e_g = e_h = e$ *(where* $\iota_\rho : \rho \cap \psi \hookrightarrow \rho$, $\iota_\psi : \rho \cap \psi \hookrightarrow \psi$*); then there is a unique* $f(g,h) : \rho \cup \psi \to \rho' \cup \psi'$ *such that* $(\rho \hookrightarrow \rho \cup \psi); f = g; (\rho' \hookrightarrow \rho' \cup \psi')$ *and* $(\psi \hookrightarrow \rho \cup \psi); f = h; (\psi' \hookrightarrow \rho' \cup \psi')$ *and furthermore* $f(g,h)$ *is a surjection as well.*

- *Let* $\rho, \psi \in Sign$, $\iota_\rho : \rho \cap \psi \hookrightarrow \rho$, $\iota_\psi : \rho \cap \psi \hookrightarrow \psi$ *and a surjection* $f : \rho \cup \psi \to (\rho \cup \psi)'$*; then there is a unique pair of surjections* $g(f) : \rho \to \rho'$, $h(f) : \psi \to \psi'$ *such that if* $\iota_\rho; g(f) = e_g; \iota_g$ *is the unique factorization of* $\iota_\rho; g(f)$ *and* $\iota_\psi; h(f) = e_h; \iota_h$ *is the unique factorization of* $\iota_\psi; h(f)$*, then* $e_g = e_h = e$ *and furthermore* $(\rho \cup \psi)' = \rho' \cup \psi'$

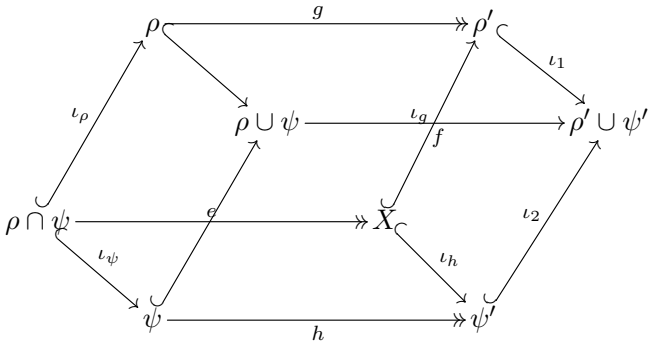

Fig. 3. Building f(g,h)

- *If* $g : \rho \to \rho'$, $h : \psi \to \psi'$ *are as above then* $g(f(g,h)) = g$ *and* $h(f(g,h)) = h$
- *If* $f : \rho \cup \psi \to (\rho \cup \psi)'$ *is as above then* $f(g(f), h(f)) = f$

Proof.

- Since *Sign* has strong inclusions, it follows that the union is a pushout of any pair of inclusions from the same object. Then let $(\iota_1, \iota_2, \rho' \cup \psi')$ be the pushout of (ι_g, ι_h) (it follows $\iota_g; \iota_1 = \iota_h; \iota_2$).

  Since $(\rho \hookrightarrow \rho \cup \psi, \psi \hookrightarrow \rho \cup \psi, \rho \cup \psi)$ is a pushout in *Sign* and $\iota_\rho; g; \iota_1 = e; \iota_g; \iota_1 = e; \iota_h; \iota_2 = \iota_\psi; h; \iota_2$, it follows there is a unique $f : \rho \cup \psi \to \rho' \cup \psi'$ such that $(\rho \hookrightarrow \rho \cup \psi); f = g; \iota_1$ and $(\psi \hookrightarrow \rho \cup \psi); f = h; \iota_2$.

  In order to prove f is a surjection, let $f = e_f; \iota_f$ be the unique factorization of f and $(\psi \hookrightarrow \rho \cup \psi); e_f = e'; \iota'$ the unique factorization of $(\psi \hookrightarrow \rho \cup \psi); e_f$. Let X be such that $e_f : \psi \cup \rho \to X$.

  Then using the properties of f, $(\psi \hookrightarrow \rho \cup \psi); f = (\psi \hookrightarrow \rho \cup \psi); e_f; \iota_f = e'; \iota'; \iota_f = g; (\psi' \hookrightarrow \rho' \cup \psi')$ and since $g \in \mathcal{E}$ and due to the uniqueness of the factorization, it follows $e' = g$ and $(\psi' \hookrightarrow \rho' \cup \psi') = \iota'; \iota_f$.

  Therefore $\iota' : \psi' \to X$ and analogously we can find an inclusion $\iota'' : \rho' \to X$; since the union is a coproduct, it follows there is a unique inclusion $\iota''' : \psi' \cup \rho' \to X$; given $\iota_f : X \to \psi' \cup \rho'$ it follows $X = \psi' \cup \rho'$, therefore $\iota_f = 1_{\psi' \cup \rho'}$ and $e_f = f$, therefore f is a surjection.

- Let $g; \iota_1' = (\rho \hookrightarrow \rho \cup \psi); f$ be the unique factorization of $(\rho \hookrightarrow \rho \cup \psi); f$ and $h; \iota_2' = (\psi \hookrightarrow \rho \cup \psi); f$, the unique factorization of $(\psi \hookrightarrow \rho \cup \psi); f$ (with $g, h \in \mathcal{E}$).

  Let $(\iota_1, \iota_2, \rho' \cap \psi')$ be the pullback of $(\iota_1', \iota_2', (\rho \cup \psi)')$. Since $(\iota_\rho; g); \iota_1' = \iota_\rho; (\rho \hookrightarrow \rho \cup \psi); f = \iota_\psi; (\psi \hookrightarrow \rho \cup \psi); f = (\iota_\psi; h); \iota_2'$, it follows there is a unique $f' : \rho \cap \psi \to \rho' \cap \psi'$ such that $f'; \iota_1 = \iota_\rho; g$ and $f'; \iota_2 = \iota_\psi; h$.

  From here we can derive the unique factorizations for $\iota_\rho; g$ as $e_{f'}; (\iota_{f'}; \iota_1)$ and for $\iota_\psi; h$ as $e_{f'}; (\iota_{f'}; \iota_2)$ with the desired property (where $f' = e_{f'}; \iota_{f'}$ is the unique factorization of f')

  Let $\iota : \rho' \cup \psi' \hookrightarrow (\rho \cup \psi)'$ be the inclusion defined by the coproduct property of the union. In order to prove $(\rho \cup \psi)' = \rho' \cup \psi'$, it is enough to find an inclusion from $(\rho \cup \psi)'$ to $\rho' \cup \psi'$.

  Since as shown $\iota_\rho; g; \iota_1' = \iota_\psi; h; \iota_2'$ it follows:

  · since *Sign* has strong inclusions, $(\rho \cup \psi, \rho \hookrightarrow \rho \cup \psi, \psi \hookrightarrow \rho \cup \psi)$ is a pushout in

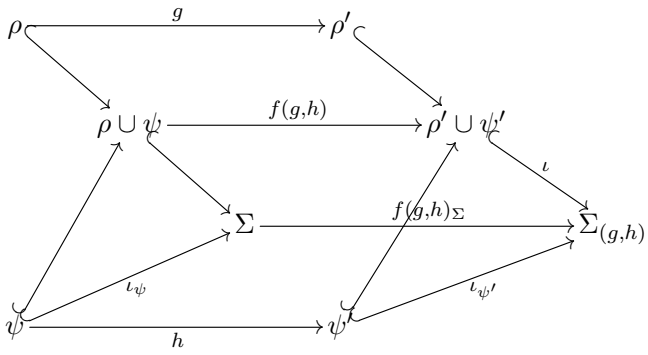

Fig. 4. Renaming

$Sign$, therefore, since by construction $g;\iota'_1 = (\rho \hookrightarrow \rho \cup \psi); f$ and $h;\iota'_2 = (\psi \hookrightarrow \rho \cup \psi); f$, f is unique with this property.

· since inclusions are monomorphisms and $(\rho' \hookrightarrow \rho' \cup \psi');\iota = \iota_1$ and $(\psi' \hookrightarrow \rho' \cup \psi');\iota = \iota_2$, it follows $\iota_\rho; g; (\rho' \hookrightarrow \rho' \cup \psi') = \iota_\psi; h; (\psi' \hookrightarrow \rho' \cup \psi')$ and given $(\rho \cup \psi, \rho \hookrightarrow \rho \cup \psi, \psi \hookrightarrow \rho \cup \psi)$ is a pushout in $Sign$, there is a unique $f'' : \rho \cup \psi \to \rho' \cup \psi'$ with $g; (\rho' \hookrightarrow \rho' \cup \psi') = (\rho \hookrightarrow \rho \cup \psi); f''$ and $h; (\psi' \hookrightarrow \rho' \cup \psi') = (\psi \hookrightarrow \rho \cup \psi); f''$ and let $f'' = e_{f''}; \iota_{f''}$ be its unique factorization.

Then

$(\rho \hookrightarrow \rho \cup \psi); f''; \iota = g; (\rho' \hookrightarrow \rho' \cup \psi'); \iota = g; \iota_1$ and analogously $(\psi \hookrightarrow \rho \cup \psi); f''; \iota = h; \iota_2$, therefore using the previous result $f = f''; \iota = e_{f''}; (\iota_{f''}; \iota)$ and since f is a surjection, it follows $\iota_{f''}; \iota$ is an identity and $\iota_{f''} : (\rho \cup \psi)' \hookrightarrow \rho' \cup \psi'$

Since $(\rho \cup \psi)' \hookrightarrow \rho' \cup \psi'$ and $\rho' \cup \psi' \hookrightarrow (\rho \cup \psi)'$, it follows $\rho' \cup \psi' = (\rho \cup \psi)'$

• The last two claims follow from the points above.

□

If the renamings on the visible and required part are consistent, then the renamed module can be retrieved from the pushout of the original hidden signature and the union of the renamed interfaces. If the signatures are sets, this means that we only rename the interfaces, and possibly symbols in the hidden signature if they clash with new names in the interfaces, the latter being added in the new hidden signature as distinct copies.

Definition 3.8 Let $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ be a module specification and $g : \rho \to \rho'$, $h : \psi \to \psi'$ surjective signature morphisms such that if $(\rho \cap \psi \hookrightarrow \rho); g = e_g; \iota_g$ and $(\rho \cap \psi \hookrightarrow \psi); h = e_h; \iota_h$ are the unique factorizations, then $e_g = e_h = e$. The *renaming* of $\mathcal{M}$ by (g, h), written $\mathcal{M} * (g, h)$, is defined as

$$\mathcal{M} * (g, h) = \rho', g(K_\rho), \psi', \Sigma_{(g,h)}, f(g, h)_\Sigma(K))$$

where $(\iota, f(g, h)_\Sigma, \Sigma_{(g,h)})$ is the pushout of $((\rho \cup \psi \hookrightarrow \Sigma), f(g, h))$.

Given the lemma and since $\rho \hookrightarrow \Sigma$, $\psi \hookrightarrow \Sigma$ and $Sign$ has pushouts that preserve inclusions, the definition is consistent.

We also note that given the assumption on the model functor, the renamed interfaces are included in the new hidden signature: since $f(g, h) : \rho \cup \psi \to \rho' \cup \psi'$,

it follows $(\rho' \hookrightarrow \rho' \cup \psi'); \iota : \rho' \hookrightarrow \Sigma_f$ and $(\psi' \hookrightarrow \rho' \cup \psi'); \iota : \psi' \hookrightarrow \Sigma_f$, therefore $\mathcal{M} * (g, h)$ is a module specification in the sense of Definition 3.1.

Equivalently, the renaming can be defined using a single morphism on the unions $f : \rho \cup \psi \to \rho'(f) \cup \psi'(f)$ (as in the lemma) as $\mathcal{M} * f = (\rho'(f), g(f)(K_\rho), \psi'(f), \Sigma_f, f_\Sigma(K))$.

The following theorem ensures that the required and visible sentences are translated consistently via renaming.

Theorem 3.9 *The renaming of a module specification is a module specification morphism (in the sense of Definition 3.5).*

Proof.

Let $\mathcal{M} * (g, h)$ be the renaming of $\mathcal{M}$ by (g, h). Then g, h are as follows, fulfilling the conditions of Definition 3.5:

- $g : (\rho, K_\rho) \to (\rho', g(K_\rho))$ is trivially a presentation morphism $(g(K_\rho) \models g(K_\rho))$;
- for $h : \psi \to \psi'$, since $h(Vth(\mathcal{M})) \subseteq Vth(\mathcal{M}')$ and $h; \iota_{\psi'} = \iota_\psi; f(g, h)_\Sigma(K)$, it follows
$$f(g, h)_\Sigma((K \cup \iota_\rho(K_\rho))^\bullet) \subseteq (f(g, h)_\Sigma(K) \cup \iota_{\rho'}(g(K_\rho)))^\bullet \tag{5}$$
- since $g; \iota_{\rho'} = \iota_\rho; f(g, h)$ and also given the commutativity in the pushout diagram, it follows from here that
$$f(g, h)_\Sigma(K) \cup \iota_{\rho'}(g(K_\rho)) \models f(g, h)_\Sigma((K \cup \iota_\rho(K_\rho))^\bullet) \tag{6}$$

□

The following theorem establishes the semantics of renaming, showing the connection between the semantics of the initial module and the semantics of the module obtained by means of renaming.

Theorem 3.10 *Given $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ a module specification, $\mathcal{M} * (g, h)$ its renaming by g and h and $m \in Mod(\mathcal{M} * (g, h))$:*

$$m \models \mathcal{M} * (g, h) \Rightarrow m \restriction_h \models \mathcal{M}$$

If additionally the inclusions are conservative, $m \models \mathcal{M} * (g, h)$ iff $m \restriction_h \models \mathcal{M}$

Proof.

(i) $\Rightarrow$

Assume $m \models \mathcal{M} * (g, h)$. Then $m \models Vth(\mathcal{M} * (g, h))$ and there is $m' \in Mod(\Sigma_{(g,h)})$ with $m' \restriction_{\psi'} = m$ and $m' \restriction_{\rho'} \models g(K_\rho)$.

Using Theorem 3.9 above and Definition 3.5, it follows $m \restriction_h \models Vth(\mathcal{M})$.

Given $g; \iota_{\rho'} = \iota_\rho; f(g, h)$, $h; \iota_{\psi'} = \iota_\psi; f(g, h)$ and the commutativity in the pushout square, it follows $m' \restriction_{f(g,h)_\Sigma}$ is an expansion of $m \restriction_h$ such that its reduct to ρ satisfies the assumed theorems.

Therefore $m \restriction_h \models \mathcal{M}$.

(ii) $\Leftarrow$

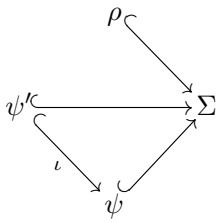

Fig. 5. Hiding

Assume $m \restriction_h \models \mathcal{M}$ and inclusions are conservative. Then there is $m' \in Mod(\Sigma_{(g,h)})$ such that $m' \restriction_{\psi'} = m$.

Then (see Figure 4), it follows $m' \restriction_{f(g,h)_\Sigma} \restriction_{\iota_\psi} = m \restriction_h \models \iota_\psi^{-1}((K \cup \iota_\rho(K_\rho))^\bullet)$ whence, given aforementioned commutative squares, $m \models Vth(\mathcal{M} * (g,h))$

□

Let us go back to the `Xor` example at the beginnig of the section. The following module can be used to define symmetric difference on sets:

```
Xor * (Bool |-> Set, or |-> union, and |-> intersect,
       not |-> complement, xor |-> sminus)
```

3.2.2 Hiding

As previously stated, a *hiding* operations retracts a part of the visible signature back into the hidden signature. For example, one may define natural numbers on top of integers by first retracting the substraction operation, renaming and then adding a total order on numbers and some properties of the numbers with respect to this order. This translated as a hiding, followed by a renaming and an enriching.

Definition 3.11 Given a module specification $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ and $\iota : \psi' \hookrightarrow \psi$, then the *hiding* is defined as the module specification

$$\psi' \square \mathcal{M} = (\rho, K_\rho, \psi', \Sigma, K)$$

Notice that the inclusion is from the new visible signature to the old one, since we are removing symbols from the latter. Back to the `Xor` example,

```
sort Bool□Xor
```

hides the `xor` operator.

Theorem 3.12 *Given $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ a module specification and $\psi' \square \mathcal{M}$ the module specification induced by hiding (where $\iota : \psi' \hookrightarrow \psi$), the pair $(1_\rho, \iota) : \psi' \square \mathcal{M} \to \mathcal{M}$ is a module specification morphism.*

Proof.

The conclusions follows since 1_ρ is trivially a presentation morphism and, given $\iota_{\psi'} = \iota; \iota_\psi$ (see Figure 5), it follows $\iota(\iota_{\psi'}^{-1}(\Gamma)) \subseteq \iota_\psi^{-1}(\Gamma)$, therefore $\iota(Vth(\psi' \square \mathcal{M})) \subseteq Vth(\mathcal{M})$ (where $\Gamma = (K \cup \iota_\rho(K_\rho))^\bullet$). □

The semantics of hiding is as expected, the models of the new module are the

reducts (via inclusions) of the models of the original module. Intuitively, the interpretations of the hidden symbols are deleted.

Theorem 3.13 *Given $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ a module specification, $\psi'\square\mathcal{M}$ the module specification induced by hiding $(\iota : \psi' \hookrightarrow \psi)$ and $m \in Mod(\mathcal{M})$, then*

$$m \models \mathcal{M} \Rightarrow m \restriction_\iota \models \psi'\square\mathcal{M}$$

If additionally the inclusions are conservative, $m \models \mathcal{M}$ iff $m \restriction_\iota \models \psi'\square\mathcal{M}$.

Proof.

(i) $\Rightarrow$

Let m' be an expansion of m. Then m' is also an expansion of $m \restriction_\iota$ (see Figure 5).

Then, if $m = m' \restriction_\psi \models Vth(\mathcal{M})$, also $m \restriction_\iota \models Vth(\mathcal{M})$ and since it has an expansion, it follows $m \restriction_\iota \models \mathcal{M}$.

(ii) $\Leftarrow$

Assume $m \restriction_\iota \models \mathcal{M}$. Then $m \models Vth(\mathcal{M})$.

If the inclusions are conservative, then there is an expansion m' of m and since $m \models Vth(\mathcal{M})$, it follows $m' \restriction_\rho \models K_\rho$.

□

Note: It might be worth investigating the effects of relaxing the restrictions on a module specifications, i.e. 'hiding' symbols and/or sentences from the requirements ($\rho' \hookrightarrow \rho$, $K_{\rho'} \subset K_\rho$).

As long as the visible sentences of the 'relaxed' module remain unchanged, this operation could be done consistently, however enforcing this condition is not computationally feasible.

3.2.3 Enriching

An *enriching* operation adds new elements to the existing module. Given the complexity of a module, this can mean many things. One can add, as notable cases:

- new hidden symbols, usually specifying either new behaviours of new constructs, if the symbols are also added to the visible signature, or otherwise more complex behaviours of old constructs;
- new visible symbols: if these symbols were already in the hidden signature, this operation can be seen as the reverse of hiding; if not, these new symbols are added in Σ as well, usually specifying new constructs;
- new required symbols: if needed to specify the new behaviours mentioned above;
- new required sentences for old symbols: these constrain the required constructs; one example may be an enriching from a module requiring a preorder, to a module requiring a partial order.

Definition 3.14 Given a module specification $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ and another module specification $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$ such that $\rho \hookrightarrow \rho'$, $\psi \hookrightarrow \psi'$ and

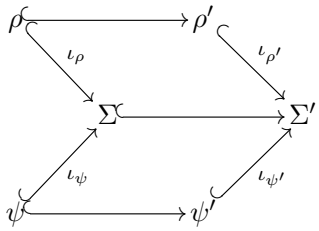

Fig. 6. Enriching

$\Sigma \hookrightarrow \Sigma'$, the *enriching* of $\mathcal{M}$ by $\mathcal{M}'$ is defined as

$$\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K') = (\rho', K_\rho \cup K_{\rho'}, \psi', \Sigma', K \cup K')$$

Theorem 3.15 *Given $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$, $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$ two module specifications such that $\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K')$ is an enriching (of $\mathcal{M}$ by $\mathcal{M}'$), the pair $(\rho \hookrightarrow \rho', \psi \hookrightarrow \psi') : \mathcal{M} \to \mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K')$ is a module specification morphism.*

Proof.

Let $\iota_1 : \rho \hookrightarrow \rho'$, $\iota_2 : \psi \hookrightarrow \psi'$ and $\iota_3 : \Sigma \hookrightarrow \Sigma'$.

Then:

- $\iota_1 : (\rho, K_\rho) \to (\rho', K_\rho \cup K_{\rho'})$ is trivially a presentation morphism (since $K_\rho \subseteq K_\rho \cup K_{\rho'}$ thus $K_\rho \subseteq (K_\rho \cup K_{\rho'})^\bullet$);
- $\iota_2(Vth(\mathcal{M})) \subseteq Vth(\mathcal{M}')$ or $\iota_2(\iota_\psi^{-1}((K \cup \iota_\rho(K_\rho))^\bullet)) \subseteq \iota_{\psi'}^{-1}((K \cup K' \cup \iota_{\rho'}(K_\rho \cup K_{\rho'}))^\bullet)$: □

Predictably, the models of the enriched module are expansions of the models of the original module.

Theorem 3.16 *Given $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$, $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$ two module specifications such that $\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K')$ is an enriching (of $\mathcal{M}$ by $\mathcal{M}'$) and $m \in Mod(\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K'))$, then*

$$m \models \mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K')$$

implies

$$m \restriction_\psi \models \mathcal{M}$$

If additionally the inclusions are conservative,

$$m \models \mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K')$$

iff

$$m \restriction_\psi \models \mathcal{M}$$

Proof.

(i) $\Rightarrow$

Let m' be an expansion of m to Σ'. Then $m' \restriction_\Sigma$ is an expansion of $m \restriction_\psi$ as $(m' \restriction_\Sigma) \restriction_\psi = (m' \restriction_{\psi'}) \restriction_\psi = m \restriction_\psi$.

Assume $m \models Vth(\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K'))$. Then it follows

$$m' \restriction_\Sigma \models K \cup \iota_\rho^{-1}(K_\rho) \tag{7}$$

therefore

$$m \restriction_\psi \models Vth(\mathcal{M}) \tag{8}$$

(ii) $\Leftarrow$

If the inclusions are conservative, then m has an expansion m' to Σ'. Furthermore, $m' \restriction_\Sigma$ is an expansion of $m \restriction_\psi$.

Then we only need to prove that if $m \restriction_\psi \models Vth(\mathcal{M})$, then $m \models Vth(\mathcal{M} * (add\ \ \rho', K_{\rho'}, \psi', \Sigma', K'))$, which can be done analogously to the previous case. □

Note: These are some particular cases of enriching that might arise in practice:

- *Basic enriching* The enriching morphism is $(1_\rho, \psi \hookrightarrow \psi')$ and $K_{\rho'} = \emptyset$; this is the case where new elements are added and specified completely, without adding anything else to the required signature ρ. In the case of the `Xor` example, a case of basic enriching is

```
Xor * (add
       exports op xnor: Bool Bool -> Bool
       rule X:Bool xnor Y:Bool => (X and Y) or (not X and not Y))
```

- *Constraining* The enriching morphism is $(\rho \hookrightarrow \rho', 1_\psi)$; this is the case where new elements are required to define the module, with no effect on the output. One further particular case is the one where the requirement inclusion is an identity, i.e. only new sentences are required (therefore the term for this kind of enriching); In the case of the `Xor` example, a case of constraining is requiring an additional property of the boolean operations

```
Xor * (add requires eq not (X:Var or Y:Var) = not X and not Y)
```

- *Completeness preserving enriching* New elements are added both to the required signature and the exported signature; this can be seen as a single inclusion (of intersections), wherefrom the new interface and requirements can be deduced (as pushouts).

3.2.4 Aggregation

Aggregation means combining two modules into one. The signatures and sentences of the resulting module are unions of the respective signatures and sentences of the initial modules.

Definition 3.17 Given two module specifications $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ and $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$, their *aggregation* is defined as

$$\mathcal{M} + \mathcal{M}' = (\rho \cup \rho', K_\rho \cup K_{\rho'}, \psi \cup \psi', \Sigma \cup \Sigma', K \cup K')$$

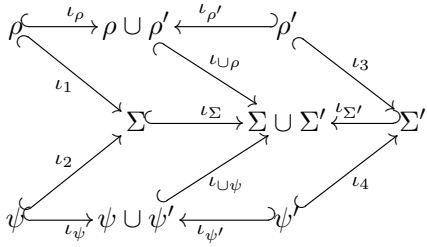

Fig. 7. Aggregation

This definition is correct since by definition of coproducts there is a unique inclusion $\iota_{\cup\rho} : \rho \cup \rho' \hookrightarrow \Sigma \cup \Sigma'$ and a unique inclusion $\iota_{\cup\psi} : \psi \cup \psi' \hookrightarrow \Sigma \cup \Sigma'$ such that:

- $\iota_\rho; \iota_{\cup\rho} = \iota_1; \iota_\Sigma$
- $\iota_\psi; \iota_{\cup\psi} = \iota_2; \iota_\Sigma$
- $\iota_{\rho'}; \iota_{\cup\rho} = \iota_3; \iota_{\Sigma'}$
- $\iota_{\psi'}; \iota_{\cup\psi} = \iota_4; \iota_{\Sigma'}$

(see also Figure 7) .

Furthermore, this operation is a simple module morphism, where the signature morphisms are inclusions, also translating visible theorems to visible theorems.

Theorem 3.18 *Given two module specifications $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ and $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$ and their aggregation $\mathcal{M} + \mathcal{M}'$, the pairs (ι_ρ, ι_ψ) and $(\iota_{\rho'}, \iota_{\psi'})$ are module specification morphisms.*

Proof.

For (ι_ρ, ι_ψ):

- ι_ρ is trivially a presentation morphism from (ρ, K_ρ) to $(\rho \cup \rho', K_\rho \cup K_{\rho'})$ (since $K_\rho \cup K_{\rho'} \models K_\rho$)
- $\iota_\psi(Vth(\mathcal{M})) \subseteq Vth(\mathcal{M} + \mathcal{M}')$ or $\iota_\psi(\iota_2^{-1}((K \cup \iota_1(K_\rho))^\bullet)) \subseteq \iota_{\cup\psi}^{-1}((K \cup K' \cup \iota_{\cup\rho}(K_\rho \cup K_{\rho'}))^\bullet)$ as follows :

$$
\begin{aligned}
&\iota_\psi(\iota_2^{-1}((K \cup \iota_1(K_\rho))^\bullet)) \\
&= \iota_{\cup\psi}^{-1}(\iota_\Sigma((K \cup \iota_1(K_\rho))^\bullet))(\text{given the commutative diagram}) \\
&= \iota_{\cup\psi}^{-1}((\iota_\Sigma(K \cup \iota_1(K_\rho)))^\bullet) \\
&= \iota_{\cup\psi}^{-1}((\iota_\Sigma(K) \cup \iota_\Sigma(\iota_1(K_\rho)))^\bullet)(\text{since } Sen \text{ preserves inclusions}) \\
&= \iota_{\cup\psi}^{-1}((\iota_\Sigma(K) \cup \iota_{\cup\rho}(\iota_\rho(K_\rho)))^\bullet) \\
&\subseteq \iota_{\cup\psi}^{-1}((K \cup K' \cup \iota_{\cup\rho}(K_\rho \cup K_{\rho'}))^\bullet)
\end{aligned}
$$

The proof for $(\iota_{\rho'}, \iota_{\psi'})$ is analogous. □

Going further to the semantics, it is easy to retrieve models of the initial modules, as reducts of a model of an aggregated module. However, we need the model functor

to preserve coproducts in order to be able to define models of the aggregated module, starting from models of the inital module.

Theorem 3.19 *Given two module specifications* $\mathcal{M} = (\rho, K_\rho, \psi, \Sigma, K)$ *and* $\mathcal{M}' = (\rho', K_{\rho'}, \psi', \Sigma', K')$*, their aggregation* $\mathcal{M} + \mathcal{M}'$*, and* $m \in Mod(\mathcal{M} + \mathcal{M}')$*, then*

$$m \models \mathcal{M} + \mathcal{M}' \text{ iff } m \restriction_\psi \models \mathcal{M} \text{ and } m \restriction_{\psi'} \models \mathcal{M}'$$

Proof.

- $\Rightarrow$ Assume $m \models \mathcal{M} + \mathcal{M}'$
  Then $m \restriction_\psi \models \mathcal{M}$ as follows:
  $m \models \mathcal{M} + \mathcal{M}'$ implies $m \models Vth(\mathcal{M} + \mathcal{M}')$ and given Theorem 3.18 this implies $m \restriction_\psi \models Vth(\mathcal{M})$ thus $m \restriction_\psi \models \mathcal{M}$.
  Furthermore, let m' be an expansion of m to $\Sigma \cup \Sigma'$. Then

$$\begin{aligned}(m' \restriction_{\iota_\Sigma}) \restriction_{\iota_2} &= m' \restriction_{\iota_2;\iota_\Sigma} \\ &= m' \restriction_{\iota_\psi;\iota_{\cup\psi}} \\ &= (m' \restriction_{\iota_{\cup\psi}}) \restriction_{\iota_\psi} \\ &= m \restriction_\psi\end{aligned}$$

  thus $m' \restriction_{\iota_\Sigma}$ is an expansion of $m \restriction_\psi$ to Σ.
  Analogously $m \restriction_{\psi'} \models \mathcal{M}'$.
- Assume $m \restriction_\psi \models \mathcal{M}$ and $m \restriction_{\psi'} \models \mathcal{M}'$.
  Let m' be an expansion of $m \restriction_\psi$ to Σ and m'', an expansion of $m \restriction_{\psi'}$ to Σ', i.e. $m' \restriction_{\iota_2} = m \restriction_\psi$ and $m'' \restriction_{\iota_4} = m \restriction_{\psi'}$. Given the hypothesis, we have that

$$m' \models K \cup \iota_1(K_\rho)$$

  and

$$m'' \models K' \cup \iota_3(K_{\rho'}) \tag{9}$$

  From this we can find an expansion of m using the model amalgamation property.
  Let $\iota_{\cap\psi} : \psi \cap \psi' \hookrightarrow \psi$, $\iota_{\cap\psi'} : \psi \cap \psi' \hookrightarrow \psi'$, $\iota_{\cap\Sigma} : \Sigma \cap \Sigma' \hookrightarrow \Sigma$, $\iota_{\cap\Sigma'} : \Sigma \cap \Sigma' \hookrightarrow \Sigma'$ as in Figure 8.
  Since the intersections are coproducts in the category of inclusions, given the pair $(\iota_{\cap\psi}; \iota_2, \iota_{\cap\psi'}; \iota_4)$, it follows that there is a unique inclusion ι (see Figure 8) such that $\iota; \iota_{\cap\Sigma} = \iota_{\cap\psi}; \iota_2$ and $\iota; \iota_{\cap\Sigma'} = \iota_{\cap\psi'}; \iota_4$.

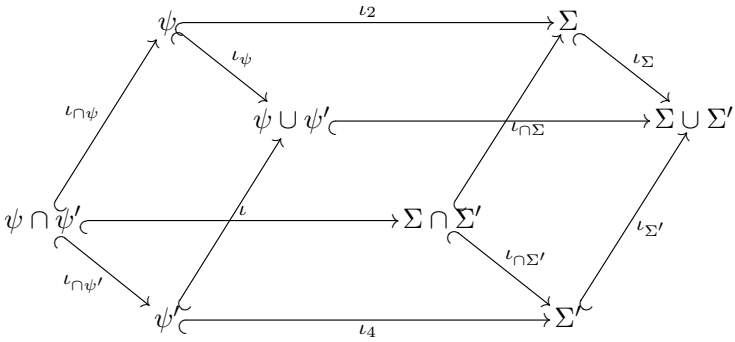

Fig. 8. Finding an expansion of m

Then:

$$\begin{aligned} m' \restriction_{\iota_{\cap\psi};\iota_2} &= (m' \restriction_{\iota_2}) \restriction_{\iota_{\cap\psi}} \\ &= (m \restriction_{\psi}) \restriction_{\iota_{\cap\psi}} \\ &= m \restriction_{\iota_\psi;\iota_{\cap\psi}} \\ &= m \restriction_{\iota_{\psi'};\iota_{\cap\psi'}} \\ &= (m \restriction'_{\psi}) \restriction_{\iota_{\cap\psi'}} \\ &= (m'' \restriction_{\iota_4}) \restriction_{\iota_{\cap\psi'}} \\ &= m'' \restriction_{\iota_{\cap\psi'};\iota_4} \end{aligned}$$

and since $(\Sigma \cup \Sigma', \iota_\Sigma, \iota_{\Sigma'})$ is a pushout in the category of inclusions (of pair $(\iota_{\cap\psi}; \iota_2, \iota_{\cap\psi'}; \iota_4)$), it follows using the model amalgamation property that there is a unique model $m_0 \in Mod(\Sigma \cup \Sigma')$ such that $m_0 \restriction_\Sigma = m'$ and $m_0 \restriction_{\Sigma'} = m''$.

Then,

$$\begin{aligned} (m_0 \restriction_{\iota_{\cup\psi}}) \restriction_{\iota_\psi} &= (m_0 \restriction_{\iota_\Sigma}) \restriction_{\iota_2} \\ &= (m_0 \restriction_{\Sigma}) \restriction_{\iota_2} \\ &= m' \restriction_{\iota_2} \\ &= m \restriction_{\iota_\psi} \end{aligned}$$

and analogously $(m_0 \restriction_{\iota_{\cup\psi}}) \restriction_{\iota_{\psi'}} = m \restriction_{\iota_{\psi'}}$. But since $\psi \cup \psi'$ is a coproduct, and Mod preserves coproducts (i.e. for any models $m_1 \in Mod(\psi)$, $m_2 \in Mod(\psi')$, there is a unique model $m_3 \in Mod(\psi \cup \psi')$ such that $m_3 \restriction_\psi = m_1$ and $m_3 \restriction_{\psi'} = m_2$) it follows that $m_0 \restriction_{\iota_{\cup\psi}} = m$, thus m_0 is an expansion of m.

Furthermore, from $m_0 \restriction_{\iota_\Sigma} = m' \models K \cup \iota_1(K_\rho)$ and $m_0 \restriction_{\iota_{\Sigma'}} = m'' \models K' \cup \iota_3(K_{\rho'})$, it follows, on one hand,

$$m_0 \models K \cup K' \tag{10}$$

and on the other

$$m_0 \models \iota_\Sigma(\iota_1(K_\rho)) = \iota_{\cup\rho}(\iota_\rho(K_\rho)) \tag{11}$$

and analogously

$$m_0 \models \iota_{\cup\rho}(\iota_{\rho'}(K_{\rho'})) \tag{12}$$

thus

$$m_0 \models \iota_{\cup\rho(K_\rho \cup K_{\rho'})} \tag{13}$$

thus

$$m_0 \models (K \cup K' \cup \iota_{\cup\rho}(K_\rho \cup K_{\rho'}))^\bullet \tag{14}$$

and since $m_0 \restriction_{\iota_{\cup\Sigma}} = m$, it follows

$$m \models \iota^{-1}_{\cup\Sigma}((K \cup K' \cup \iota_{\cup\rho}(K_\rho \cup K_{\rho'}))^\bullet) = Vth(\mathcal{M} + \mathcal{M}') \tag{15}$$

thus $m \models \mathcal{M} + \mathcal{M}'$.

□

4 Conclusions and Future Work

We have defined abstract modules and module operations intended to be used for the $\mathbb{K}$ framework; apart from the constructs and behaviours it defines - as a signature and set of sentences, an abstract module specifies which of those constructs it exports and what it requires other modules to export. Module operations defined and analyzed here are: renaming, hiding, enriching and aggregation.

The $\mathbb{K}$ institution is assumed to be an inclusive institution with strong inclusions and pushouts which preserve inclusions; additionally, the model functor is required to preserves pushouts and coproducts. Formally defining the $\mathbb{K}$ institution, as well as proving that these properties hold for the instititution, are the subject of future work.

Other directions of future work are analyzing parametrization and some properties of the module system, as well as implementing the module system in the $\mathbb{K}$ tool.

References

[1] Roşu, Grigore, and Traian Florin Şerbănuţă , *An Overview of the* $\mathbb{K}$ *Semantic Framework*, Journal of Logic and Algebraic Programming **79** (2010), 397–434

[2] Diaconescu, Răzvan, Joseph Goguen, and Petros Stefaneas, *Logical support for modularisation*, Logical Environments (1993), Cambridge University Press, 83–130

[3] Goguen, Joseph and Grigore Rosu, *Composing Hidden Information Modules over Inclusive Institutions*, From Object-Orientation to Formal Methods, LLNCS **2635** (2004), Springer Berlin / Heidelberg, 96–123

[4] Rosu, Grigore, *The Institution of order-sorted equational logic*, Bulletin of EATCS, **53** (1994)

[5] Borzyszkowski, Tomasz, *Higher-Order Logic and Theorem Proving for Structured Specifications*, in Algebraic Development Techniques, (WADT 99), LNCS (1999), Springer, 401–418

[6] Goguen, Joseph A. and Rod M. Burstall, *Institutions: abstract model theory for specification and programming*, J. ACM. **39** (1992), 95–146.

[7] Roşu, Grigore, *Abstract semantics for module composition*, Technical Report CSE2000–0653, University of California at San Diego, May 2000. Written August 1997.

[8] Goguen, Joseph, and Will Tracz, *An Implementation-Oriented Semantics for Module Composition*, Foundations of component-based systems (2000), Leavens, Gary T. and Sitaraman, Murali, 231–263

[9] Hills, Mark, and Grigore Roşu, *Towards a Module System for* $\mathbb{K}$, Recent Trends in Algebraic Development Techniques (WADT'08), LNCS **5486** (2008), 187–205

[10] Borovansky, Peter, Claude Kirchner, Helene Kirchner, Pierre-etienne Moreau, and Christophe Ringeissen, *An Overview of ELAN*, ENTCS **15** (1998), 55–70

[11] Bergstra, Jan A., Jan Heering, and Paul Klint, *Module Algebra*, J. ACM **37** (1990), 335–372

[12] Durán, Francisco, and José Meseguer, *Maude's module algebra*, Sci. Comput. Program. **66** (2007), 125–153

[13] Astesiano, Egidio, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki, *CASL: the Common Algebraic Specification Language*, Theor. Comput. Sci. **286** (2002), 153–196

[14] Mac Lane, Saunders, "Categories for the Working Mathematician", Springer, 1971

[15] Diaconescu, Razvan, "Institution-independent Model Theory", Birkhäuser, 2008