Math. Struct. in Comp. Science (1993), *vol.* 11, *pp.* 1–000

Very Abstract Specifications: A Formalism Independent Approach†

MAURA CERIOLI and GIANNA REGGIO

Dipartimento di Informatica e Scienze dell'Informazione
Via Dodecaneso 35
16146 Genova (Italy)
email: {cerioli, reggio}@disi.unige.it

Received

Two operations are presented for a modular approach to the definition of frameworks for rigorous development of software, formally represented as institutions.
The first one generalizes models, allowing them to have more structure than the minimal required by their declared signatures, as it happens for software modules, having local routines that do not appear in their interface.
The second one extends sentences, and their interpretation in models, allowing sentences on richer signatures to be used as formulae for poorer ones.
Combining the application of these operations, powerful institutions can be defined, like those for *very abstract entities*, or for *hyper-loose algebraic specifications*.
The compatibility of different sequential applications of these operations and properties of the resulting institutions are studied as well.

1. Introduction

In the last twenty years, several different formal approaches have been proposed supporting rigorous software development. Even restricting the attention only to *algebraic* frameworks, the variations are many, depending on the signatures (from the original homogeneous ones, to many-sorted, order-sorted, higher-order, polymorphic, with or without predicates and so on), the sentences (from pure equations without variables to full first-order, or even higher-order, formulae), the models (from total, to partial, non-strict, state-based algebras) and the satisfaction relation (from standard evaluation semantics, to observational definitions).

The proliferation of frameworks, is in a sense unavoidable, because a formalism providing tools to deal with all possible software features, for instance convenient for the sequential, concurrent and object-oriented paradigms, and in all development phases, from the requirement to the design, if any, would be a *monster* and would become out of date in a short time.

But, on the other side, having so many possibilities is confusing, especially for the naive

† This work has been partially supported by the projects MURST-40% Modelli e Specifiche di Sistemi Concorrenti and Modelli della Computazione e dei Linguaggi di Programmazione.

users, and makes the choice of the *best* paradigm, for any given problem, quite difficult. Moreover, the expertise accumulated during a project development may be useless for the next specification task, because changing the problems, the needed formalism changes as well. Finally, it is often the case that, in order to provide some new features, brand new frameworks, or adaptations of already existing ones, have to be developed almost from scratch, proving standard properties, with standard techniques, but in several slightly different settings.

In our opinion, a solution for, or at least an improvement of, this situation is to have means to modularly build the formalisms themselves, assembling pieces of already well-known formalisms, or tuning them, by adding only the "local" features. Indeed, in this way the theory can be worked out once and forever and, even more important, the expertise acquired by the end users can be reused. Moreover, since building a framework in this way is easy and does not require a time consuming development of new theories, it becomes convenient to select for any given problem the best formalism, having all the features needed by the particular problem under examination, but as simple as possible.

In this paper, following the well-established approach by Goguen and Burstall, see e.g. (Burstall and Goguen, 1984; Burstall and Goguen, 1992), specification frameworks are formalized as *institutions*. Thus, enrichments and assembling of formalisms become, in this setting, operations among institutions.

The need for such a modular approach to the formalism construction has already been sporadically addressed in the literature. Consider for instance the *duplex institutions* in (Burstall and Goguen, 1992), where an institution is built whose sentences come from two input institutions, also applied in the database field in (Reichwein and Fiadeiro, 1992). Another example is the *extension by universal closure* in (Sannella and Tarlecki, 1988), of a given institution and a set of its signature morphisms, where sentences are enriched along these signature morphisms regarding the extra-symbols as variables universally quantified. Moreover the *institution of implementation specifications* of (Beierle and Voss, 1987) enriches an institution by tools to deal with implementation.

Here we face the problem of enriching an institution in a way that models possibly have more structure than the minimal required by their signatures, as it happens for software modules, having local routines that do not appear in their interface. Thus, in this new setting we could characterize larger classes of models, having not all the same syntax, but sharing a minimal structure. Moreover, sentences are extended to provide the capability of expressing properties on the possible local functionalities, stating properties not only on the models (i.e. about the interpretations), but also on the syntaxes themselves (i.e. about the actual structure of the models).

Concrete instances of this constructions can be found in the description of institutions for *very abstract specifications* in the field of concurrency, like for instance the very abstract entity specifications in (Reggio, 1991), the very abstract entity specifications with temporal logic in (Astesiano and Reggio, 1993a), the very abstract entity specifications with event logic in (Astesiano and Reggio, 1993b; Reggio, 1993), each one in several variants, like with first-order, conditional, equational logic, with partial, non-strict, generalized models and so on.

Another application of the very abstract operation, in the field of abstract data types, is the proof that the hyper-loose algebraic specifications in (Pepper, 1991), whose models on Σ are Σ'-algebras on some Σ' "extending" Σ, is an institution. A specification in this institution describes classes of algebras sharing a common syntax and satisfying properties on such common part, but with possibly some more structure, analogously to software realizations of a module, that are allowed to have, besides the operations required by the interface, further internal operations. However the sentences in this institution are the same as in the parameter institution, so that properties on the syntax of the specifications cannot be imposed. Extending also the sentences, here we get the institution of very abstract data types, which supports the specification of high-level requirements on modules also about their interfaces (e.g. constraints either in the number of operations or on the number of arguments of the operations, due to limits of the admissible implementations).

Let us consider, now, the institution of entity algebras, see (Reggio, 1991), providing a formal framework for algebraic specifications of concurrent systems, where some signature operations are used to explicitly describe the concurrent structure (i.e. to define the system components, both static and dynamic, and the system architecture). Thus very abstract specifications (built on the entity institution) describe classes of entity algebras on possibly different signatures, i.e. formal models of systems with possibly different concurrent structures, satisfying common properties. Moreover, these examples have alread been used in some industrial case studies of the specification, at different levels of abstraction, of a substation for the electric power distribution (see (Reggio et al., 1992)).

The very abstract operation is modularly described as the composition of two basic operations on institutions: $\mathsf{ABSTRACT}$, that abstracts the models on a signature Σ, by regarding as abstract Σ-models the actual models on each "extension" of Σ, and EXTEND, that extends the set of sentences, so that formulas about signature properties are allowed (but this operation is far more general and can be used, for example, to add in a uniform way logical operators, e.g. the equality).

The arguments of $\mathsf{ABSTRACT}$ are an institution and a family of signature "extensions", that are signature morphisms satisfying some technical conditions. Thus, as the other parameters only depend on the signature category, the proof of the existence of such parameters can be shared by all institutions with the same syntactic part and, in particular, by the result of $\mathsf{ABSTRACT}$ itself. Therefore if two possible extensions are available for the same (signature) institution, it is possible to apply sequentially both constructions. As a paramount instance of $\mathsf{ABSTRACT}$ application, here we show the construction of signature extensions for the many-sorted signatures (with predicates), so that the same construction can be used for most "algebraic" institutions (e.g. institutions with partial or non-strict models and with every logic).

As it may be expected, the result of an application of $\mathsf{ABSTRACT}$ is strictly related to its argument. Indeed, though in the result the very abstract models on a given signature are more than in the original institution, they can be canonically flattened to (standard) models, simply forgetting the extra structure they may have. On the converse, each standard model is obviously also a very abstract model, that happen to have no local structure. This relationships is formalized by saying that institution morphisms and maps of institutions relate the input to the output of the operation in both ways. Thus, morphisms and maps between institutions that are used as inputs for ABSTRACT may be lifted to work on the corresponding results, as well.

While abstracting models only involves the signature part of an institution, to extend sentences we require that each (extensible) signature is associated with an (intuitively) richer signature, whose sentences will be used as extended sentences for the starting signature. Moreover, each model is naturally expanded to a model of the extended signature, so that the validity of the new sentences can be easily defined. Therefore, EXTEND affects both sentences and validity relation and requires information on how to extend signatures and models.

Quite natural applications for the EXTEND operation are the construction of second-order logic starting from a first-order institution, where each signature is extended by functional sorts, that are interpreted in each extended model as the corresponding function spaces, and the definition of observational specification institutions (see Section 3). The latter case shows that, interpreting the definition of observational satisfaction as an application of EXTEND, the required verification for the result to be an institution simplify to check on the arguments of EXTEND.

Moreover, as extensively shown in Section 4, EXTEND can be used in connection with ABSTRACT, enriching very abstract institutions, and in particular that built starting from many-sorted logic, by the expressive capability for requiring the actual syntax of the very abstract models to satisfy some properties. This is achieved by introducing sorts and operations for the syntactical elements of a signature, using an internalization principle.

Several possible choices of sentence extension are available. Here we have adopted a simple, but sufficiently expressive one, presenting not only the resulting framework, but also specifications of relevant data types, made within such formalism, to prove that it is convenient.

The paper is organized as follows. In Section 2 we consider the problem of building a new institution by abstracting the models w.r.t. the syntax of another one, while Section 3 is devoted to the problem of extending the sentences of an institution to get another one. Finally in Section 4 we describe the very abstract institutions by combining the two operations previously defined.

A shorter version of this paper, presenting only the core ideas, has been published in (Cerioli and Reggio, 1994).

2. Abstracting Models w.r.t. Syntax

In mathematical practice it is quite common to regard algebraic structures, like fields or rings, as poorer structures, like groups or monoids. This can be interpreted from two points of view; the first intuition is that we *forget* about the extra structure, so that if we have a ring, then we also have a group on the same set and with the same sum, inverse and zero as the ring. Thus, different rings with the same underline group result in one group and, more interestengly, we cannot for instance use the product(s) in order to prove properties on the sum. The second interpretation is that a ring in itself *is* a group, i.e. groups are all those entities that have *at least* the group operations, but can as well have more structure. Therefore the extra operations (if any) are still available to shorten proofs, clarify reasoning etc.

A more applicative example of the same situation is the definition of software modules realizing a data type. Indeed, any such module is required to associate a function with each operation of the data type, but it is quite common, in the practice, to have (private) local definitions, different for every actual module, giving to the module an extra-structure. Then all local operations (and types) of a module can be hidden by some encapsulating interface, so that users cannot use them anymore. This corresponds, in the previous example, to forgetting the ring structure. But it is also reasonable to export (some of) the local operations, in order to improve complexity of algorithms or simplify termination proofs and such. In other words, the module itself, with all its structure, is regarded as a realization of the data type.

Using the concept of *institution*, see e.g. (Burstall and Goguen, 1984; Burstall and Goguen, 1992), to represent logical frameworks, the "forget/hide" viewpoint is immediately available, since it corresponds to the use of the *reduct* functors, that for any change of syntax represent how models have to be translated. But the notion that a richer structure should be regarded in itself, without translations, as a poorer structure too, cannot be immediately represented. In order to describe this point of view we define an operation ABSTRACT that applied to a logical framework $\mathcal{I}$ yields a framework over $\mathcal{I}$, where models of a syntax are required to provide a semantic counterpart for all elements of the syntax, but can have some extra-structure, i.e. where models are, for each signature, the models of the original institution on "extensions" of such signature.

Def. 2.1. An *institution* $\mathcal{I}$ (see e.g. (Burstall and Goguen, 1984)) consists of a category $\mathbf{Sign}$ of *signatures*, a functor $Sen\colon \mathbf{Sign} \to \mathbf{Set}$ giving the set of *sentences* over a signature, a functor $Mod\colon \mathbf{Sign} \to \mathbf{Cat}^{\mathrm{Op}}$ giving the category of *models* on a signature, and a *satisfaction relation* $\models \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$ for each Σ object in $\mathbf{Sign}$, sometimes denoted by $\models_\Sigma$, such that for each morphism $\phi\colon \Sigma \to \Sigma'$ in $\mathbf{Sign}$, the *satisfaction condition*

$$M' \models_{\Sigma'} Sen(\phi)(\xi) \iff Mod(\phi)(M') \models_\Sigma \xi$$

holds for each M' in $|Mod(\Sigma')|$ and each ξ in $Sen(\Sigma)$. □

In the sequel we assume that $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ is an institution and discuss the elements needed in order to build an institution $\mathcal{H} = \mathsf{ABSTRACT}(\mathcal{I}, \ldots)$, with the same syntax as $\mathcal{I}$ (signature and sentences), but whose models are allowed to have some extra structure.

2.1. *Abstract Models on* Σ

Intuitively in $\mathcal{H}$ a signature Σ represents the minimal structure that its models have, but the models can have a richer structure than the one explicitly described by Σ. Thus, the Σ-models in $\mathcal{H}$ are the Σ'-models in $\mathcal{I}$, for some Σ' "extending" Σ. In most examples signatures are structured (families of) sets, so that extensions are simply set-inclusions and hence correspond to a particular subclass of signature (mono)morphisms. This leads to consider the class of these morphisms, called *admissible*, as one of the ABSTRACT parameters. Note that two minimal requirements have to be imposed on this class: that the identities are admissible, corresponding to the intuition that each signature is the trivial extension of itself, and that the class of admissible morphisms is closed under composition, because extending an extension should result in an extension, too. Therefore, it is natural to formalize the admissible morphisms as the arrows of a subcategory having the same object class as the signature category.

Def. 2.2. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution; then a family of *admissible morphisms* for $\mathcal{I}$ is any subcategory $\mathbf{HMon}$ of $\mathbf{Sign}$ s.t. $|\mathbf{HMon}| = |\mathbf{Sign}|$.
Here and in the sequel we write $m: \Sigma \hookrightarrow \Sigma'$ to denote that m is an admissible morphism from Σ into Σ', that is $m \in \mathbf{HMon}(\Sigma, \Sigma')$. Moreover, we simply write $A_{|m}$ for $Mod(m)(A)$. Given $\Sigma, \Sigma' \in |\mathbf{Sign}|$, we say that Σ' *extends* Σ iff there exists an admissible morphism in $\mathbf{HMon}(\Sigma, \Sigma')$. □

A class of admissible morphisms available for each institution consists of the identities. But, since in this case the construction of the very abstract models in the sequel collapses to the identity, this class is useless.

A more interesting class of admissible morphisms is that of monomorphisms, as they are composable and include identities. This choice corresponds more closely to the intuition of extension we want to capture and indeed, in the following we will see a motivating example using monomorphisms as admissible morphisms.

Using admissible morphisms to represent signature extensions, the abstract models on any signature Σ are pairs $\langle A, m \rangle$, where A is a (standard) model on a signature Σ' extending Σ via m, that is $m: \Sigma \hookrightarrow \Sigma'$ and $A \in |Mod(\Sigma')|$. Note that we need to keep track of the way Σ' extends Σ, because in general Σ' may be an extension of Σ in different ways, as several morphisms with the same domain and codomain can be admissible.

Let us consider, now, the arrows between these new models, in order to get a category. Since abstract models are pairs, also a morphism between two such models, say from $\langle A, m_1: \Sigma \hookrightarrow \Sigma' \rangle$ into $\langle B, m_2: \Sigma \hookrightarrow \Sigma'' \rangle$, is a pair of arrows between the corresponding components. The second element is an arrow from m_1 into m_2 (seen as objects of the comma category $\Sigma \downarrow \mathbf{HMon}$, i.e. an admissible morphism $m: \Sigma' \hookrightarrow \Sigma''$ in $\mathbf{HMon}$ s.t. the following diagram commutes

$$\begin{tikzcd} & \Sigma \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\ \Sigma' \arrow[rr, "m"'] & & \Sigma'' \end{tikzcd}$$

Thus, if such an m exists, B is an algebra on an extension of the actual signature of A and hence it is natural to choose as first component of the model morphism, a Σ-morphism from A into $B_{|m}$, preserving all the structure of A and not only the minimal required by Σ.

Def. 2.3. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and $\mathbf{HMon}$ a family of admissible morphisms for $\mathcal{I}$.

For each $\Sigma \in |\mathbf{Sign}|$, the category $HMod(\Sigma)$ is defined by:

objects: $\{\ \langle A, m\rangle \mid m \in \mathbf{HMon}(\Sigma, \Sigma'), A \in |Mod(\Sigma')| \text{ and } \Sigma' \in |\mathbf{Sign}|\ \}$;
morphisms: $HMod(\Sigma)(\langle A, m_1 \colon \Sigma \hookrightarrow \Sigma'\rangle, \langle B, m_2 \colon \Sigma \hookrightarrow \Sigma''\rangle) =$
$\{\langle p, m\rangle \mid m \in \mathbf{HMon}(\Sigma', \Sigma''), m \cdot m_1 = m_2 \text{ and } p \in Mod(\Sigma')(A, B_{|m})\}$;
identities: $Id_{\langle A, m: \Sigma \hookrightarrow \Sigma'\rangle} = \langle Id_A, Id_{\Sigma'}\rangle$;
composition: $\langle q, m' \colon \Sigma'' \hookrightarrow \Sigma'''\rangle \cdot \langle p, m \colon \Sigma' \hookrightarrow \Sigma''\rangle = \langle q_{|m} \cdot p, m' \cdot m\rangle$. □

A graphical view of this composition is given below, where $\langle p, m\rangle \colon \langle A, m_1\rangle \to \langle B, m_2\rangle$ and $\langle q, m'\rangle \colon \langle B, m_2\rangle \to \langle C, m_3\rangle$.

$$\begin{tikzcd}
& Mod(\Sigma) & \\
A \xrightarrow{p} B_{|m} \xrightarrow{q_{|m}} C_{|m' \cdot m} \in Mod(\Sigma') \arrow[ur, "Mod(m_1)"] & B \xrightarrow{q} C_{|m'} \in Mod(\Sigma'') \arrow[l, "Mod(m)"] \arrow[u, "Mod(m_2)"] & C \in Mod(\Sigma''') \arrow[l, "Mod(m')"] \arrow[ul, "Mod(m_3)"']
\end{tikzcd}$$

Notice that if $\mathbf{HMon}(\Sigma, \Sigma') = \emptyset$ for all $\Sigma \neq \Sigma'$ and $\mathbf{HMon}(\Sigma, \Sigma) = \{Id_\Sigma\}$, then the construction of $HMod(\Sigma)$ yields (an isomorphic copy of) $Mod(\Sigma)$ itself.

Prop. 2.4. For every $\Sigma \in |\mathbf{Sign}|$, $HMod(\Sigma)$ is a category.

Proof. It is trivial to check that "$\cdot$" is associative in $HMod(\Sigma)$ and that $Id_{\langle A, m\rangle}$ is the identity of the composition. □

There is (at least) another natural choice of the model morphisms in $HMod(\Sigma)$, that is to have as morphisms from $\langle A, m_1 \colon \Sigma \hookrightarrow \Sigma'\rangle$ into $\langle B, m_2 \colon \Sigma \hookrightarrow \Sigma''\rangle$ the Σ-morphisms in $Mod(\Sigma)$ from $A_{|m_1}$ into $B_{|m_2}$. However, in this way the morphisms do not depend on the extra-structure of Σ' and Σ''. Hence two abstract models, say $\langle A, m\rangle$ and $\langle B, m\rangle$, on the same extension can be isomorphic if their restrictions along m are such, while A and B are not even related by a homomorphism either way. Thus, unmotivated identities among the models would be introduced.

Instead, following our choice, two abstract models are isomorphic in the new institution iff in the starting institution they are models on isomorphic signatures (i.e. the structure of the first is a renaming of the structure of the second one) and (their renamed structures) are isomorphic, accordingly with the intuition that the nature of the specified models is the same and in the new institution we are only able to specify "bigger" classes of original models.

Prop. 2.5. Using the notation of Definition 2.3, for all morphisms $\langle p, m\rangle$ in $HMod(\Sigma)$, we have that $\langle p, m\rangle$ is an isomorphism iff both m and p are isomorphisms and the inverse of m is admissible.

Proof. Straightforward check of the definition. □

The models on a signature Σ can be naturally regarded as very abstract models on that signature, seen as extension of itself by the identity, and the very abstract models on a signature can be translated into models on the signature, by the reduct associated with their second component, in a sort of *flattening*.

Def. 2.6. Using the notation of Definition 2.3, let $Emb\colon Mod(\Sigma) \to HMod(\Sigma)$ be the functor defined by:

on objects: for each $A \in |Mod(\Sigma)|$, $Emb(A) = \langle A, Id_\Sigma \rangle$;
on arrows: for each $p \in Mod(\Sigma)(A, B)$, $Emb(p) = \langle p, Id_\Sigma \rangle$.

Moreover, let $Flat\colon HMod(\Sigma) \to Mod(\Sigma)$ be the functor defined by:

on objects: for each $\langle A, m \rangle \in |HMod(\Sigma)|$, $Flat(\langle A, m \rangle) = A_{|m}$;
on arrows: for each $\langle p, m \rangle \in Mod(\Sigma)(\langle A_1, m_1 \rangle, \langle A_2, m_2 \rangle)$, $Flat(\langle p, m \rangle) = p_{|m_1}$. □

It is straightforward to check that both *Emb* and *Flat* are functors; moreover they are adjoint to each other, so that *Emb* is a coreflexive subcategory inclusion.

Prop. 2.7. Using the notation of Definition 2.6, *Emb* is the left adjoint to *Flat* and for every $\langle A, m \rangle \in |HMod(\Sigma)|$ the counit of the adjunction is $\epsilon_{\langle A,m \rangle} = \langle Id_{(A_{|m})}, m \rangle$.

Proof. Let us consider an object $\langle A, m \rangle \in |HMod(\Sigma)|$ and show that for each $B \in |Mod(\Sigma)|$ and each $\langle p, m' \rangle\colon Emb(B) \to Emb \cdot Flat(\langle A, m \rangle)$ in $HMod(\Sigma)$ the unique $q\colon B \to Flat(\langle A, m \rangle)$ s.t. $\epsilon_{\langle A,m \rangle} \cdot Emb(q) = \langle p, m' \rangle$ is p itself.
Since $\langle p, m' \rangle$ is a morphism in $HMod(\Sigma)$ from $Emb(B) = \langle B, Id_\Sigma \rangle$ into $\langle A, m \rangle$, $m' \cdot Id_\Sigma = m$, i.e. $m' = m$, and p is a morphism in $Mod(\Sigma)$ from B into $A_{|m}$, that is $p\colon B \to Flat(\langle A, m \rangle)$.
Moreover, $\epsilon_{\langle A,m \rangle} \cdot Emb(p) = \langle Id_{(A_{|m})}, m \rangle \cdot \langle p, Id_\Sigma \rangle = \langle Id_{(A_{|m})_{|Id_\Sigma}} \cdot p, m \cdot Id_\Sigma \rangle = \langle p, m \rangle$.
Thus the following diagram commutes.

$$\begin{tikzcd}
 & \begin{array}{c} Emb \cdot Flat(A) = \\ \langle (A_{|m}), Id_\Sigma \rangle \end{array} \arrow[dl, "\begin{array}{c}\epsilon_{\langle A,m \rangle} = \\ \langle Id_{(A_{|m})}, m \rangle\end{array}"'] & \begin{array}{c} Flat(\langle A, m \rangle) = \\ A_{|m} \end{array} \\
\langle A, m \rangle & & \\
 & \begin{array}{c} Emb(B) = \\ \langle B, Id_\Sigma \rangle \end{array} \arrow[ul, "\langle p\colon A \to B_{|m}, m \rangle"] \arrow[uu, "\begin{array}{c}Emb(p) = \\ \langle p, Id_\Sigma \rangle\end{array}"'] & B \arrow[uu, "p"']
\end{tikzcd}$$

Finally if $\epsilon_{\langle A,m \rangle} \cdot Emb(q) = \langle p, m \rangle$, then $\langle Id_{(A_{|m})}, m \rangle \cdot \langle q, Id_\Sigma \rangle = \langle p, m \rangle$, i.e. $\langle Id_{(A_{|m})_{|Id_\Sigma}} \cdot q, m \cdot Id_\Sigma \rangle = \langle q, m \rangle = \langle p, m \rangle$ and hence $q = p$. □

2.2. *Translating Abstract Models along Signature Morphisms*

Algebraic approaches to the semantics of specifications regard as very relevant the notion of model translation (or reduct) along a signature morphism, because it allows to abstract from the name of the operations of a module. This is reflected, in the institution language, by the functorial nature of the model component and by the satisfaction condition, that formalizes the slogan "*truth is invariant under change of notation*".

Thus, we have to define the translation of very abstract models along signature morphisms, generalizing the definition of $HMod(\Sigma)$ to a functor from $\mathbf{Sign}$ to $\mathbf{Cat}^{\mathrm{Op}}$; in other words, a family of functors $HMod(\phi): HMod(\Sigma_2) \to HMod(\Sigma_1)$, for every $\phi \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$, has to be defined, preserving identities and composition.

Let us fix a signature morphism $\phi: \Sigma_1 \to \Sigma_2$; then $HMod(\phi): HMod(\Sigma_2) \to HMod(\Sigma_1)$ should transform a model $\langle A, m: \Sigma_2 \hookrightarrow \Sigma'_2 \rangle$ in a pair $\langle B, e: \Sigma_1 \hookrightarrow \Sigma'_1 \rangle$, consisting of an extension of the signature Σ_1 and a model on such extension. Moreover, if ϕ can be extended to a morphism ϕ' between Σ'_1 and Σ'_2, then the model B can be easily defined as the translation of A along ϕ'.

Therefore we need a uniform way of building extensions of $\phi: \Sigma_1 \to \Sigma_2$ starting from any extension $m: \Sigma_2 \hookrightarrow \Sigma'_2$ of its codomain. Graphically the situation is the following, where we know the continuous arrows and have to determine the dashed lines.

$$
\begin{tikzcd}
\Sigma_1 \arrow[r, "\phi"] \arrow[d, dashed, "e"'] & \Sigma_2 \arrow[d, "m"] \\
\Sigma'_1 \arrow[r, dashed, "\phi'"'] & \Sigma'_2
\end{tikzcd}
$$

Notice that, given a class of admissible morphisms, there are different sensible choices for building extensions of Σ_1 starting from the Σ_2-extensions. Indeed, let us consider as admissible all monomorphisms (but the example works as well for the set theoretic inclusions as class of admissible morphisms in the case of standard algebraic institutions). Then if we pick a (non-identical) monomorphism ϕ we can choose ϕ itself as e and the identity as ϕ' or, vice versa, the identity as e and ϕ itself as ϕ'.

Thus, in the following definition we require that *local backward extensions* are selected satisfying the minimal conditions sufficient for each $HMod(\phi)$, defined by translation of models along such ϕ''s, to be a functor.

Def. 2.8. Let $\mathbf{HMon}$ be a class of admissible morphisms for an institution $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$. A *local backward extension* on $\mathbf{HMon}$ for $\mathcal{I}$, consists of a signature $\mathtt{sig}(\phi, m)$, a morphism $\mathtt{mor}(\phi, m): \mathtt{sig}(\phi, m) \to \Sigma'_2$ and an admissible morphism $\mathtt{mon}(\phi, m): \Sigma_1 \hookrightarrow \mathtt{sig}(\phi, m)$ for each signature morphism $\phi: \Sigma_1 \to \Sigma_2$ and each admissible morphism $m: \Sigma_2 \hookrightarrow \Sigma'_2 \in \mathbf{HMon}$ satisfying the following conditions †:

1 The following diagram commutes, i.e. $m \cdot \phi = \mathtt{mor}(\phi, m) \cdot \mathtt{mon}(\phi, m)$:

† Note that the (a)-conditions follow from the corresponding (b)-conditions (and/or from the (c)-conditions) and are mentioned for the sake of clearness.

$$
\begin{tikzcd}
\Sigma_1 \arrow[r, "\phi"] \arrow[d, "\mathtt{mon}(\phi, m)"'] & \Sigma_2 \arrow[d, "m"] \\
\mathtt{sig}(\phi, m) \arrow[r, "\mathtt{mor}(\phi, m)"'] & \Sigma_2'
\end{tikzcd}
$$

2 The choice of $\mathtt{sig}$, $\mathtt{mor}$ and $\mathtt{mon}$ is natural w.r.t. the second argument:

(a) $\mathtt{sig}(\phi, m' \cdot m) = \mathtt{sig}(\mathtt{mor}(\phi, m), m')$;

(b) $\mathtt{mor}(\phi, m' \cdot m) = \mathtt{mor}(\mathtt{mor}(\phi, m), m')$;

(c) $\mathtt{mon}(\phi, m' \cdot m) = \mathtt{mon}(\mathtt{mor}(\phi, m), m') \cdot \mathtt{mon}(\phi, m)$;

$$
\begin{tikzcd}
\Sigma_1 \arrow[rr, "\phi"] \arrow[d, "\mathtt{mon}(\phi, m)"'] \arrow[ddr, "\mathtt{mon}(\phi, m' \cdot m)"] & & \Sigma_2 \arrow[d, "m"] \arrow[ddr, "m' \cdot m"] & \\
\mathtt{sig}(\phi, m) \arrow[rr, "\mathtt{mor}(\phi, m)"] \arrow[dr, "\mathtt{mon}(\mathtt{mor}(\phi, m), m')"'] & & \Sigma_2' \arrow[dr, "m'"] & \\
& \begin{array}{c}\mathtt{sig}(\mathtt{mor}(\phi, m), m') \\ = \mathtt{sig}(\phi, m' \cdot m)\end{array} \arrow[rr, "\begin{array}{c}\mathtt{mor}(\phi, m' \cdot m) = \\ \mathtt{mor}(\mathtt{mor}(\phi, m), m')\end{array}"'] & & \Sigma_2''
\end{tikzcd}
$$

3 The identity as second argument is preserved:

(a) $\mathtt{sig}(\phi, Id_{\delta_1(\phi)}) = \delta_0(\phi)$;

(b) $\mathtt{mor}(\phi, Id_{\delta_1(\phi)}) = \phi$;

(c) $\mathtt{mon}(\phi, Id_{\delta_1(\phi)}) = Id_{\delta_0(\phi)}$. □

There are two obvious candidates for local backward extension, that are $\mathtt{mor}(\phi, m) = m \cdot \phi$ (and accordingly $\mathtt{mon}(\phi, m) = Id_{\delta_0(\phi)}$) and $\mathtt{mor}(\phi, m) = Id_{\delta_1(\phi)}$ (and accordingly $\mathtt{mon}(\phi, m) = m \cdot \phi$). The former actually satisfies all conditions required by the definition of local backward extension, though methodologically it does not have much sense, because the extension is trivial and, as we will see in the sequel, it does not satisfy the conditions required to get an institution. The latter is not well defined, in general, because $m \cdot \phi$ is not required to be admissible for any choice of admissible morphisms; for instance if admissible morphisms are all monomorphisms in **Sign** and **Sign** contains at least a morphism ϕ that is not mono, then $\mathtt{mon}(\phi, Id_{\delta_1(\phi)}) = \phi$ is not admissible. But even if all morphisms are admissible, this choice does not satify condition 3c if ϕ is not an identity, because $\mathtt{mon}(\phi, Id_{\delta_1(\phi)}) = \phi$.

The conditions required from a local backward extension, besides the technical needs in the proofs of functoriality for $HMod(\phi)$, are determined by the intuition that a local backward extension along $\phi\colon \Sigma_1 \to \Sigma_2$ and $m\colon \Sigma_2 \hookrightarrow \Sigma'_2$ should endow Σ_1 with the algebraic structure present in Σ'_2 that is not already present in Σ_2. Indeed, let us consider again our motivating example of software modules. We start from some representation of modules in a standard algebraic framework, the institution $\mathcal{I}$, and want to regard the signature of a module as its visible syntax, allowing local internal operations and hence we (adopt a notion of signature extension and) build *HMod*. Then, it is natural to require the translation of module along a renaming of its visible signature to yield the module itself with the visible part accordingly renamed and the local structure unchanged as far as possible. Indeed, changing the types of the visible part affects the functions having parameter(s) or result of some global type.

Application 2.9. Let us consider many-sorted first-order logic with equality. Thus, each signature Σ consists of a set S of *sorts*, an $S^* \times S$-indexed family of *function symbols* and an S^+-indexed family of *predicate symbols*. Function symbols are used to build terms and predicate symbols applied to terms yield atomic sentences.
Signature morphisms are consistent renaming of symbols; thus if a function expects an argument of sort s, then its translation requires an argument of the image of s along the signature morphism.
As admissible morphisms, **YMon**, we consider the plain inclusions between many-sorted signatures, i.e.:

$$\mathbf{YMon}_{\Sigma,\Sigma'} = \{\iota\colon \Sigma \hookrightarrow \Sigma' \mid \iota(x) = x \text{ for all symbols } x\}.$$

Then, for instance, let us consider the signature Σ_1 of (finite) sets of natural numbers with a sort representing (discrete) time.

sig Σ_1 =
sorts *nat*, *time*, *set*
opns $0\colon \to nat$
$Reset\colon \to time$
$\emptyset\colon \to set$
$S\colon nat \to nat$
$Ins\colon nat \times set \to set$
$Clock\colon time \to time$

Moreover, let us consider its implementation by integers and lists, using integers to represent both natural numbers and time and with sets described by lists. This is formalized by the following signature Σ_2

sig Σ_2 =
sorts *int*, *list*
opns $Zero\colon \to int$
$\lambda\colon \to list$
$Inc, Dec\colon int \to int$
$Push\colon int \times list \to list$

and by the signature morphism ϕ defined by

$$\begin{array}{lclclclclcl}
\phi(nat) & = & int & \quad & \phi(0) & = & Zero & \quad & \phi(S) & = & Inc \\
\phi(time) & = & int & & \phi(Reset) & = & Zero & & \phi(Clock) & = & Inc \\
\phi(set) & = & list & & \phi(\emptyset) & = & \lambda & & \phi(Ins) & = & Push
\end{array}$$

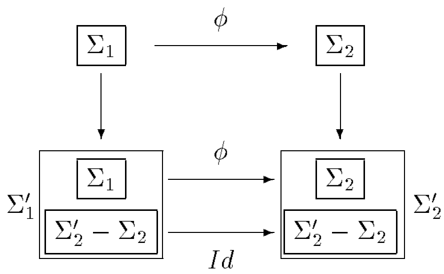
Let us finally consider an extension of Σ_2, where a boolean sort is added

sig $\Sigma_2' =$
enrich Σ_2 **by**
sorts *bool*
opns *True*, *False*: $\to$ *bool*
Isempty: *list* $\to$ *bool*

Then the expected extension of Σ_1 is:

sig $\Sigma_1' =$
enrich Σ_1 **by**
sorts *bool*
opns *True*, *False*: $\to$ *bool*
Isempty: *set* $\to$ *bool*

More in general, the intuition behind the definition of the backward extension of Σ_1 is to add to Σ_1 all components of $\Sigma_2' - \Sigma_2$, as it is graphically represented below.



However, this solution is too simplified to cope with more sophisticated examples. Let us see some instances of the problems that can be encountered, to get an intuition of the possible solutions. In the sequel, we will use *local* for symbols belonging to an extension but not to the visible part of the signature.

Let us consider again the previous example, but consider the following extensions Σ_2'', with local functions having visible sorts in their arity.

sig $\Sigma_2'' =$
enrich Σ_2 **by**
sorts *bool*
opns *True*, *False*: $\to$ *bool*
Isempty: *list* $\to$ *bool*
Tail: *list* $\to$ *list*
Head: *list* $\to$ *int*

In this case, as *list* represents *set*, we can imagine that the arguments of sort *list* in the backward copies of *Tail* and *Head* should be replaced by arguments of sort *set*. But, as *int* stands for both *nat* and *time*, we should expect two backward copies of *Head*, one for each possible replacement of *int* by corresponding sorts form Σ_1. Thus the intended extension of Σ_1 is:

sig $\Sigma_1'' =$
enrich Σ_1 **by**
sorts *bool*
opns $True,\ False\colon \to bool$
$Isempty\colon set \to bool$
$Tail\colon set \to set$
$Head\colon set \to nat$
$Head\colon set \to time$

where we can use the same symbol for both copies of the *Head*, taking advantage of the overloading of function and predicate symbols granted by the definition of functions as an indexed family instead than a set equipped with an arity function.
Accordingly to our intuition that the source is replacing the target signature in the extension to get the backward extension, the elements of the target signature that are not in the image of the morphism are dropped. Therefore, local function (predicate) symbols having dropped sorts in their arity have to disappear too, as in the following case. Let us consider again the starting example, but regarding ϕ as a morphism from Σ_1 into Σ_2'', that is as its composition with the embedding of Σ_2 into Σ_2'', and as extension the following signature

sig $\Sigma_2''' =$
enrich Σ_2'' **by**
$IsOrdered\colon list \to bool$
$Sort\colon list \to list$

In this case, as *bool* does not belong to the image of the morphism, and hence cannot be replaced by any sort of Σ_1, nor is a local symbol, and hence we do not have to add it to the backward extension, it will disappear. Accordingly the local function *IsOrdered* has to be dropped. Thus the intended extension of Σ_1 is:

sig $\Sigma_1''' =$
enrich Σ_1 **by**
$Sort\colon set \to set$

The last problem we want to illustrate is how to deal with name clashes. Let us consider again the starting example, but using as extension the following signature

sig $\Sigma_2^{IV} =$
enrich Σ_2 **by**
$S\colon int \to int$

If we naively apply the technique suggested at the beginning, then we have two incarnations of S in the backward extension, one originated from the enrichment by the local structure from Σ_2^{IV} and the otherone already present in the source. Unfortunately, as for each fixed arity the function symbols form a set, this leads to an unduely identification of the two incarnations. Therefore, a new symbol for the "local" S has to be provided and the intended extension of Σ_1 is:

sig $\Sigma_1^{IV} =$
enrich Σ_1 **by**
$S'\colon nat \to nat$

Thus, from this case analysis, we have that the simplified picture we proposed has to be generalized to account for

— non injective signature morphisms, inducing duplications of local functions (predicates) with the same name, but different arity or result type;
— non surjective signature morphisms, possibly discarding sorts and hence making local functions (predicates) symbols to be dropped if some discarded global sort appears in their arity;
— name clashes between the local symbols in the extension and visible symbols in the source signature, requiring to introduce new names to avoid unduely identifications.

The last case actually is the unique problematic from a technical viewpoint, as it seems to be impossible to find a uniform way of introducing new symbols. Therefore, we have to move to *abstract* signatures, that is for each isomorphism class of signatures we arbitrary choose a representative. In order to be sure that we are not discarding needed admissible morphisms, we fix many details of the representatives for isomorphic signatures that we are using. Notice that the reduction from concrete to (a particular choice of) abstract signatures does not affect the specification language built over an institution, but is analogous to translation of user defined identifiers to their internal (usually disambiguated) representation made through key tables and does not show up at the user level (though it is used to define the semantics of the specifications defined by the user).
In order to keep the presentation as simple as possible, and since such restriction does not seem too severe from a practical viewpoint, we stick to *finite* signatures, that is from now on we take into account only signatures with a finite number of sorts, operations and predicates.
Let us fix a denumerable universe of sorts $\mathcal{U}_S$, with enumeration function $\mathsf{srt}\colon \mathbb{N} \to \mathcal{U}_S$[‡], a denumerable universe of operation symbols $\mathcal{U}_O$, with enumeration function $\mathsf{opn}\colon \mathbb{N} \to \mathcal{U}_O$ and a denumerable universe of predicate symbols $\mathcal{U}_P$, with enumeration function $\mathsf{prd}\colon \mathbb{N} \to \mathcal{U}_P$. Then an *abstract signature* $\Sigma = (S, OP, PR)$ is a first-order signature s.t.

— $S = \{\mathsf{srt}(i) \mid 1 \leq i \leq n\}$ for some $n \in \mathbb{N}$;
— for each $w \in S^*$ and $s \in S$ there exists $m \in \mathbb{N}$ s.t. $OP_{w,s} = \{\mathsf{opn}(i) \mid 1 \leq i \leq m\}$;
— for each $w \in S^+$ there exists $k \in \mathbb{N}$ s.t. $PR_w = \{\mathsf{prd}(i) \mid 1 \leq i \leq k\}$;

Let **FOESign** be the category of abstract signatures, and let $\mathcal{FOE} =$ $(\mathbf{FOESign}, \mathit{FOESen}, \mathit{FOEMod}, \models^{FOE})$ denote the institution of many-sorted first-order logic with equality and let us consider as admissible morphisms the embedding in **FOESign**. Then, for instance, the first example is interpreted as a presentation in an algebraic language for the following abstract signatures (the "abstract" names are assigned in declaration order)

‡ In order to allow infinite signatures as well, it suffices to use an enumeration $\mathsf{srt}\colon \mathbb{N} \times \mathbb{N} \to \mathcal{U}_S$ with the restriction that $\mathsf{srt}(i, j) \in S$ implies $\mathsf{srt}(i', j') \in S$ for all $i' < i$ and all j' and that there is a maximum i s.t. $\mathsf{srt}(i, j) \in S$. Analogously for functions and predicates. However, many games on indexes have to be plaied to keep the construction straight.

$$
\begin{array}{ll}
\textbf{sig } \Sigma_1 = & \\
\textbf{sorts} & \mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3) \\
\textbf{opns} & \mathsf{opn}(1) \colon \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(2) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(3) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(1) \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(1) \times \mathsf{srt}(3) \to \mathsf{srt}(3) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(2) \to \mathsf{srt}(2)
\end{array}
$$

$$
\begin{array}{ll}
\textbf{sig } \Sigma_2 = & \\
\textbf{sorts} & \mathsf{srt}(1), \mathsf{srt}(2) \\
\textbf{opns} & \mathsf{opn}(1) \colon \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(2) \\
& \mathsf{opn}(1), \mathsf{opn}(2) \colon \mathsf{srt}(1) \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(1) \times \mathsf{srt}(2) \to \mathsf{srt}(2)
\end{array}
$$

$$
\begin{array}{ll}
\textbf{sig } \Sigma'_2 = & \\
\textbf{sorts} & \mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3) \\
\textbf{opns} & \mathsf{opn}(1) \colon \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(2) \\
& \mathsf{opn}(1), \mathsf{opn}(2) \colon \mathsf{srt}(1) \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(1) \times \mathsf{srt}(2) \to \mathsf{srt}(2) \\
& \mathsf{opn}(1),\ \mathsf{opn}(2) \colon \to \mathsf{srt}(3) \\
& \mathsf{opn}(1) \colon \mathsf{srt}(2) \to \mathsf{srt}(3)
\end{array}
$$

and, accordingly, the morphism [§] ϕ is

$$
\begin{array}{lll}
\phi(\mathsf{srt}(1)) = \mathsf{srt}(1) & \phi_{\lambda,\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi_{\mathsf{srt}(1),\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \\
\phi(\mathsf{srt}(2)) = \mathsf{srt}(1) & \phi_{\lambda,\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi_{\mathsf{srt}(2),\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \\
\phi(\mathsf{srt}(3)) = \mathsf{srt}(2) & \phi_{\lambda,\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi_{\mathsf{srt}(1)\cdot\mathsf{srt}(3),\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1)
\end{array}
$$

Then, the expected backward extension of Σ_1 is Σ_1 itself enriched by the elements of Σ'_2 that do not belong to Σ_2. Notice that, since sort sets of abstract signatures are given by enumeration of intervals, the extra elements are of the form $\{\mathsf{srt}(i) \mid n_2 < i \leq n'_2\}$, where n_2 is the cardinality of the sorts of the target signature and n'_2 of its extension, and have to be translated, so that the first element, $\mathsf{srt}(n_2 + 1)$ gets the next free index in the source signature, that is $\mathsf{srt}(n_1 + 1)$, where n_1 is the cardinality of the sorts of the source signature. Therefore, the representative of a local sort $\mathsf{srt}(i)$, with $i > n_2$, will be the sort $\mathsf{srt}(i + (n_2 - n_1))$ and analogously for function and predicate symbols. Thus, for instance, in this example we have to add a sort to represent $\mathsf{srt}(3)$ to the sort set of Σ_1, that is $\{\mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3)\}$ and hence we must add $\mathsf{srt}(4)$ and associate $\mathsf{srt}(4)$ with $\mathsf{srt}(3)$ by the extension of ϕ.
Therefore, we get as extension

$$
\begin{array}{ll}
\textbf{sig } \Sigma'_1 = & \\
\textbf{sorts} & \mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3), \mathsf{srt}(4) \\
\textbf{opns} & \mathsf{opn}(1) \colon \to \mathsf{srt}(1) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(2) \\
& \mathsf{opn}(1) \colon \to \mathsf{srt}(3)
\end{array}
$$

[§] Notice that, due to the heavy overloading of function symbols in abstract signatures, to describe morphisms it is needed a decoration to distinguish the several components dealing with function symbols w.r.t. the arity.

$\mathsf{opn}(1){:}\, \mathsf{srt}(1) \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1){:}\, \mathsf{srt}(1) \times \mathsf{srt}(3) \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1){:}\, \mathsf{srt}(2) \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1),\ \mathsf{opn}(2){:} \rightarrow \mathsf{srt}(4)$
$\mathsf{opn}(1){:}\, \mathsf{srt}(3) \rightarrow \mathsf{srt}(4)$

and, accordingly, the extended morphism $\mathtt{ymor}(\phi, em)$, denoted by ϕ', is

$$\begin{array}{lll}
\phi'(\mathsf{srt}(1)) = \mathsf{srt}(1) & \phi'_{\lambda,\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi'_{\mathsf{srt}(1),\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \\
\phi'(\mathsf{srt}(2)) = \mathsf{srt}(1) & \phi'_{\lambda,\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi'_{\mathsf{srt}(2),\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \\
\phi'(\mathsf{srt}(3)) = \mathsf{srt}(2) & \phi'_{\lambda,\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi'_{\mathsf{srt}(1)\cdot\mathsf{srt}(3),\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \\
\phi'(\mathsf{srt}(4)) = \mathsf{srt}(3) & \phi'_{\lambda,\mathsf{srt}(4)}(\mathsf{opn}(1)) = \mathsf{opn}(1) & \phi'_{\lambda,\mathsf{srt}(4)}(\mathsf{opn}(2)) = \mathsf{opn}(2) \\
 & & \phi'_{\mathsf{srt}(3),\mathsf{srt}(4)}(\mathsf{opn}(1)) = \mathsf{opn}(1)
\end{array}$$

Let $\phi{:}\, \Sigma_1 \rightarrow \Sigma_2$, where $\Sigma_1 = (S_1, OP_1, PR_1)$ and $\Sigma_2 = (S_2, OP_2, PR_2)$, be the morphism

$$(\sigma{:}\, S_1 \rightarrow S_2, \{\psi_{w,s}{:}\, (OP_1)_{w,s} \rightarrow (OP_2)_{\sigma(w),\sigma(s)}\}_{w\in S^*, s\in S}, \{\pi_w{:}\, (PR_1)_w \rightarrow (PR_2)_{\sigma(w)}\}_{w\in S^+}),$$

using σ to denote its pointwise extension to strings, too, in **FOESign** and $em{:}\, \Sigma_2 \hookrightarrow \Sigma'_2$, where $\Sigma'_2 = (S'_2, OP'_2, PR'_2)$, be an admissible morphism in **YMon**.
Here and in the sequel, we will use $|X|$ to denote the cardinality of a set X.

— $\mathtt{ysig}(\phi, em) = (S, OP, PR)$, where:
  * $S = \{\mathsf{srt}(i) \mid 1 \leq i \leq n\}$ for $n = n_1 + n'_2 - n_2$, where $n_1 = |S_1|$, $n'_2 = |S'_2|$, and $n_2 = |S_2|$. In the following we will denote by $\overline{\sigma}$ the extension of σ to S^*, defined by $\overline{\sigma}(s_1 \ldots s_n) = \overline{\sigma}(s_1) \ldots \overline{\sigma}(s_n)$ and
  $$\overline{\sigma}(s) = \begin{cases} \sigma(s) & \text{if } s \in S_1 \\ \mathsf{srt}(i + n_2 - n_1) & \text{if } s = \mathsf{srt}(i) \text{ and } i > n_1 \end{cases}$$
  * for each $w \in S^*$, $s \in S$, $OP_{w,s} = \{\mathsf{opn}(i) \mid 1 \leq i \leq m\}$ for $m = m_1 + m'_2 - m_2$, where $m_1 = |(OP_1)_{w,s}|$, $m'_2 = |(OP'_2)_{\overline{\sigma}(w),\overline{\sigma}(s)}|$ and $m_2 = |(OP_2)_{\overline{\sigma}(w),\overline{\sigma}(s)}|$. In the following we will denote by $\overline{\psi}$ the extension of ψ to OP, defined by
  $$\overline{\psi}_{w,s}(f) = \begin{cases} \psi_{w,s}(f) & \text{if } f \in (OP_1)_{w,s} \\ \mathsf{opn}(i + m_2 - m_1) & \text{if } f = \mathsf{opn}(i) \text{ and } i > m_1 \end{cases}$$
  * for each $w \in S^+$, $PR_w = \{\mathsf{prd}(i) \mid 1 \leq i \leq k\}$ for $k = k_1 + k'_2 - k_2$, where $k_1 = |(PR_1)_w|$, $k'_2 = |(PR'_2)_{\overline{\sigma}(w)}|$ and $k_2 = |(PR_2)_{\overline{\sigma}(w)}|$. In the following we will denote by $\overline{\pi}$ the extension of π to PR, defined by
  $$\overline{\pi}_w(p) = \begin{cases} \pi_w(p) & \text{if } p \in (PR_1)_w \\ \mathsf{prd}(i + k_2 - k_1) & \text{if } p = \mathsf{prd}(i) \text{ and } i > k_1 \end{cases}$$

  It is obvious to see that $\mathtt{ysig}(\phi, em)$ is a many-sorted abstract signature. Notice that dangling edges are automatically taked care of, as deleted sorts do not appear in S and hence the corresponding set is not taken into account.

— $\mathtt{ymon}(\phi, em)$ is the inclusion of Σ_1 into $\mathtt{ysig}(\phi, em)$, that is, $\mathtt{ymon}(\phi, em)(x) = x$ for all symbols x of the signature Σ_1.

— $\mathtt{ymor}(\phi, em): \mathtt{ysig}(\phi, em) \to \Sigma_2' = (\overline{\sigma}, \overline{\psi}, \overline{\pi})$, that is it coincides with ϕ on Σ_1 and is the obvious index translation on the symbols from $\Sigma_2' - \Sigma_2$.

Let us verify that `ysig`, `ymon` and `ymor` defined above are a local backward extension on **YMon**.

Proof of 1) We have to show that $\mathtt{ymor}(\phi, em) \cdot \mathtt{ymon}(\phi, em) = em \cdot \phi$.
Let $s \in S_1$. Then $(\mathtt{ymor}(\phi, em) \cdot \mathtt{ymon}(\phi, em))(s) = \mathtt{ymor}(\phi, em)(\mathtt{ymon}(\phi, em)(s)) = \mathtt{ymor}(\phi, em)(s) = \overline{\sigma}(s) = \sigma(s) = (em \cdot \phi)(s)$.
Analogously on operations and predicates.

Proof of 2)

2a) We have to show that $\mathtt{ysig}(\phi, em' \cdot em) = \mathtt{ysig}(\mathtt{ymor}(\phi, em), em')$, where $em': \Sigma_2'' \hookrightarrow \Sigma_2''$ is an admissible morphism in **YMon** and $\Sigma_2'' = (S_2'', OP_2'', PR_2'')$. Let us denote $\mathtt{ysig}(\phi, em' \cdot em)$ by (S, OP, PR), $\mathtt{ysig}(\mathtt{ymor}(\phi, em), em')$ by $(\overline{S}, \overline{OP}, \overline{PR})$ and $\mathtt{ysig}(\phi, em)$ by (S_1', OP_1', PR_1').
Let us check that the two signatures have the same sorts, that is, that $S = \overline{S}$. By definition of `ysig`, we have $S = \{\mathsf{srt}(i) \mid 1 \leq i \leq n\}$ for $n = n_1 + (n_2'' - n_2)$, $n_1 = |S_1|$, $n_2'' = |S_2''|$ and $n_2 = |S_2|$.
Analogously, we have $\overline{S} = \{\mathsf{srt}(i) \mid 1 \leq i \leq \overline{n}\}$ for $\overline{n} = n_1' + (n_2'' - n_2')$, $n_1' = |S_1'|$, $n_2'' = |S_2''|$ and $n_2' = |S_2'|$ and, as S_1' is the set of sorts of the signature $\mathtt{ysig}(\phi, em)$, $S_1' = \{\mathsf{srt}(i) \mid 1 \leq i \leq n_1'\}$ for $n_1' = n_1 + (n_2' - n_2)$.
Therefore, $\overline{n} = n_1 + (n_2' - n_2) + (n_2'' - n_2') = n_1 + (n_2'' - n_2) = n$ and hence $S = \overline{S}$.
Analogously it can be shown that the two signatures have the same operations and the same predicates.

2b) We have to show that $\mathtt{ymor}(\phi, em' \cdot em) = \mathtt{ymor}(\mathtt{ymor}(\phi, em), em')$; let us see that both yields the same result on each sort s of $\mathtt{ysig}(\phi, em' \cdot em)$.
Using the notation of the previous point, we have for each $s = \mathsf{srt}(i) \in S$

$$\mathtt{ymor}(\phi, em' \cdot em)(s) = \overline{\sigma}(s) = \begin{cases} \sigma(s) & \text{if } i \leq n_1 \\ \mathsf{srt}(i - n_1 + n_2) & \text{if } n_1 < i \end{cases}$$

and analogously $\mathtt{ymor}(\mathtt{ymor}(\phi, em), em')(s) =$

$$\begin{cases} \mathtt{ymor}(\phi, em)(s) & \text{if } i \leq n_1' \\ \mathsf{srt}(i - n_1' + n_2') & \text{if } n_1' < i \end{cases} = \begin{cases} \sigma(s) & \text{if } i \leq n_1 \\ \mathsf{srt}(i - n_1 + n_2) & \text{if } n_1 < i \leq n_1' \\ \mathsf{srt}(i - n_1' + n_2') & \text{if } n_1' < i \end{cases}$$

But $n_1' = n_1 + (n_2' - n_2)$; thus, $i - n_1' + n_2' = i - n_1 - n_2' + n_2 + n_2' = i - n_1 + n_2$. Therefore, the two definitions coincide.
The proof that $\mathtt{ymor}(\phi, em' \cdot em)$ and $\mathtt{ymor}(\mathtt{ymor}(\phi, em), em')$ coincide also on operations and predicates is analogous to the above proof for sorts.

2c) Trivial, as there is at the most one admissible morphism between two signatures.

Proof of 3 Let $\phi: \Sigma_1 \to \Sigma_2$, where $\Sigma_1 = (S_1, OP_1, PR_1)$ and $\Sigma_2 = (S_2, OP_2, PR_2)$. If we show that $\mathtt{ysig}(\phi, Id_{\Sigma_2}) = (S, OP, PR)$ coincides with Σ_1, then by definition of $\mathtt{ymor}(\phi, Id_{\Sigma_2})$ and $\mathtt{ymon}(\phi, Id_{\Sigma_2})$, the conditions 3b and 3c immediately follow.
But, by definition, $|S| = |S_1| + |S_2| - |S_2| = |S_1|$ and hence $S = S_1$; analogously it can be proved the wanted identification for operations and predicates. $\square$

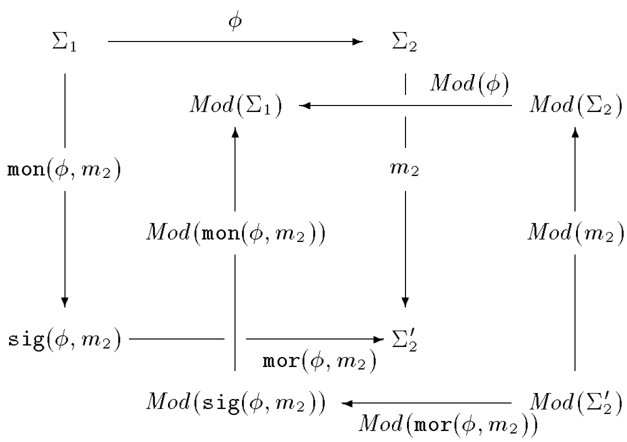
Let us now show that the properties required for local backward extensions suffice to have that each $HMod(\phi)$ is a functor.

Prop. 2.10. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and $\mathtt{sig}, \mathtt{mon}, \mathtt{mor}$ be a local backward extension on a class **HMon** of admissible morphisms for $\mathcal{I}$.
For every $\phi \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$, let $HMod(\phi): HMod(\Sigma_2) \to HMod(\Sigma_1)$ be defined by:

on objects:

$$HMod(\phi)(\langle A_2, m_2: \Sigma_2 \hookrightarrow \Sigma_2' \rangle) = \langle Mod(\mathtt{mor}(\phi, m_2))(A_2), \mathtt{mon}(\phi, m_2): \Sigma_1 \hookrightarrow \mathtt{sig}(\phi, m_2) \rangle$$

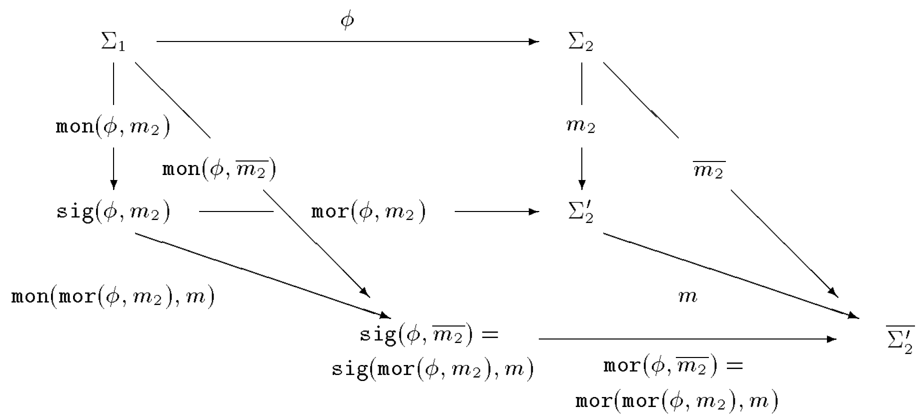
for each $\langle A_2, m_2 \rangle \in |HMod(\Sigma_2)|$, i.e. the admissible morphism m_2 is translated into the admissible morphism provided by the backward extensions and the model A is accordingly translated along the (model-interpretation of the) extension of ϕ, as the front side of the following picture shows (the back side reminds the syntactic counterpart):



on morphisms:

$$HMod(\phi)(\langle p, m \rangle) = \langle Mod(\mathtt{mor}(\phi, m_2))(p), \mathtt{mon}(\mathtt{mor}(\phi, m_2), m) \rangle$$

for each $\langle p, m \rangle \in HMod(\Sigma_2)(\langle A_2, m_2: \Sigma_2 \hookrightarrow \Sigma_2' \rangle, \langle \overline{A_2}, \overline{m_2}: \Sigma_2 \hookrightarrow \overline{\Sigma_2'} \rangle)$, accordingly with the translation of models; a complexive picture is given below;



Then $HMod(\phi)$ is a functor.

Proof. It is immediate to see that $HMod(\phi)$ sends objects of $HMod(\Sigma_2)$ into objects of $HMod(\Sigma_1)$ and that it preserves the functionality of model morphisms, i.e. that $HMod(\phi)(\langle p, m\rangle)$ is a morphism from $HMod(\phi)(\langle A, m_2\rangle)$ into $HMod(\phi)(\langle \overline{A_2}, \overline{m_2}\rangle)$ for every morphism $\langle p: A \rightarrow \overline{A_2}_{|m}, m: \Sigma_2' \hookrightarrow \overline{\Sigma_2'}\rangle$ from $\langle A, m_2: \Sigma_2 \hookrightarrow \Sigma_2'\rangle$ into $\langle \overline{A_2}, \overline{m_2}: \Sigma_2 \hookrightarrow \overline{\Sigma_2'}\rangle$. Thus it is sufficient to check that $HMod(\phi)$ preserves identities and composition.

Identities For every $\langle A_2, m: \Sigma_2 \hookrightarrow \Sigma_2'\rangle \in |HMod(\Sigma_2)|$

$$\begin{array}{lll}
HMod(\phi)(\langle Id_{A_2}, Id_{\Sigma_2'}\rangle) & = & \text{by definition} \\
\langle Mod(\mathtt{mor}(\phi, m))(Id_{A_2}), \mathtt{mon}(\mathtt{mor}(\phi, m), Id_{\Sigma_2'})\rangle & = & \text{since } Mod(\mathtt{mor}(\phi, m)) \text{ is a functor} \\
\langle Id_{Mod(\mathtt{mor}(\phi,m))(A_2)}, \mathtt{mon}(\mathtt{mor}(\phi, m), Id_{\Sigma_2'})\rangle & = & \text{by condition 3c} \\
\langle Id_{Mod(\mathtt{mor}(\phi,m))(A_2)}, Id_{\mathtt{sig}(\phi,m)}\rangle & = & \text{by definition} \\
Id_{HMod(\phi)(\langle A_2,m\rangle)} & &
\end{array}$$

Composition Let $\langle A, m_2: \Sigma_2 \hookrightarrow \Sigma_2'\rangle, \langle \overline{A_2}, \overline{m_2}: \Sigma_2 \hookrightarrow \overline{\Sigma_2'}\rangle, \langle \widehat{A_2}, \widehat{m_2}: \Sigma_2 \hookrightarrow \widehat{\Sigma_2'}\rangle$ be objects of $HMod(\Sigma_2)$, $\langle p, m\rangle$ belong to $HMod(\Sigma_2)(\langle A, m_2\rangle, \langle \overline{A_2}, \overline{m_2}\rangle)$ and $\langle q, m'\rangle$ to $HMod(\Sigma_2)(\langle \overline{A_2}, \overline{m_2}\rangle, \langle \widehat{A_2}, \widehat{m_2}\rangle)$.
Then, by definition of composition in $HMod(\Sigma_2)$, $\langle q, m'\rangle \cdot \langle p, m\rangle = \langle q_{|m} \cdot p, m' \cdot m\rangle$ and hence

$$HMod(\phi)(\langle q, m'\rangle \cdot \langle p, m\rangle) = \langle Mod(\mathtt{mor}(\phi, m_2))(q_{|m} \cdot p), \mathtt{mon}(\mathtt{mor}(\phi, m_2), m' \cdot m)\rangle.$$

Moreover, $HMod(\phi)(\langle p, m\rangle) = \langle Mod(\mathtt{mor}(\phi, m_2))(p), \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)\rangle$ and $HMod(\phi)(\langle q, m'\rangle) = \langle Mod(\mathtt{mor}(\phi, \overline{m_2}))(q), \mathtt{mon}(\mathtt{mor}(\phi, \overline{m_2}), m')\rangle$, so that $HMod(\phi)(\langle q, m'\rangle) \cdot HMod(\phi)(\langle p, m\rangle)$ is the pair whose first element is $Mod(\mathtt{mor}(\phi, \overline{m_2}))(q)_{|\mathtt{mon}(\mathtt{mor}(\phi,m_2),m)} \cdot Mod(\mathtt{mor}(\phi, m_2))(p)$ and the second is $\mathtt{mon}(\mathtt{mor}(\phi, \overline{m_2}), m') \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)$.
Let us consider just the first components.

$$\begin{array}{lll}
Mod(\mathtt{mor}(\phi, m_2))(q_{|m} \cdot p) & = & \text{since } Mod(\mathtt{mor}(\phi, m_2)) \text{ is a functor} \\
Mod(\mathtt{mor}(\phi, m_2))(q_{|m}) \cdot Mod(\mathtt{mor}(\phi, m_2))(p) & &
\end{array}$$

Therefore, it is sufficient to prove that

$$Mod(\mathtt{mor}(\phi, m_2))(q_{|m}) = Mod(\mathtt{mor}(\phi, \overline{m_2}))(q)_{|\mathtt{mon}(\mathtt{mor}(\phi,m_2),m)};$$

that is, since Mod is a functor, that $Mod(\mathtt{mor}(\phi, \overline{m_2}) \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m))$ and $Mod(m \cdot \mathtt{mor}(\phi, m_2))$ coincide. But

$$\begin{array}{lcl} Mod(\mathtt{mor}(\phi, \overline{m_2}) \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)) & = & \text{by condition 2b, as } \overline{m_2} = m \cdot m_2 \\ Mod(\mathtt{mor}(\mathtt{mor}(\phi, m_2), m) \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)) & = & \text{by condition 1} \\ Mod(m \cdot \mathtt{mor}(\phi, m_2)) & & \end{array}$$

Let us consider now the second components.

$$\begin{array}{l} \mathtt{mon}(\mathtt{mor}(\phi, \overline{m_2}), m') \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m) = \text{by condition 2b, as } \overline{m_2} = m \cdot m_2 \\ \mathtt{mon}(\mathtt{mor}(\mathtt{mor}(\phi, m_2), m), m') \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m) = \text{by condition 2c (for } \phi = \mathtt{mor}(\phi, m_2)) \\ \mathtt{mon}(\mathtt{mor}(\phi, m_2), m' \cdot m) \end{array}$$

□

2.3. *Very Abstract Institutions*

Given a local backward extension, for each signature morphism ϕ a functor $HMod(\phi)$ can be defined, translating the abstract models. But in general such construction is not compositional, as shown by the following example.

Example 2.11. Consider an alternative local backward extension for *concrete* first-order signatures where the extension is built making a (non necessarily disjoint) union between the symbols in the source signatures and those local to the extension of the target. That is, for each $\phi: \Sigma_1 \to \Sigma_2$, where $\Sigma_1 = (S_1, OP_1, PR_1)$ and $\Sigma_2 = (S_2, OP_2, PR_2)$ and each admissible $em: \Sigma_2 \hookrightarrow \Sigma'_2$, where $\Sigma'_2 = (S'_2, OP'_2, PR'_2)$, the local backward extension is defined as follows.

— $\mathtt{wsig}(\phi, em) = (S, OP, PR)$, where:
   * $S = S_1 \cup (S'_2 - S_2)$.
     In the following we will denote by $\overline{\sigma}$ the extension of σ to S^*, defined by $\overline{\sigma}(s_1 \ldots s_n) = \overline{\sigma}(s_1) \ldots \overline{\sigma}(s_n)$ and $\overline{\sigma}(s) = \begin{cases} \sigma(s) & \text{if } s \in S_1 \\ s & \text{otherwise} \end{cases}$
   * $OP_{w,s} = (OP_1)_{w,s} \cup ((OP'_2)_{\overline{\sigma}(w),\overline{\sigma}(s)} - (OP_2)_{\overline{\sigma}(w),\overline{\sigma}(s)})$, for all $w \in S^*$, $s \in S$
   * $PR_w = (PR_1)_w \cup ((PR'_2)_{\overline{\sigma}(w)} - (PR_2)_{\overline{\sigma}(w)})$, for all $w \in S^+$.

— $\mathtt{wmon}(\phi, em)$ is the inclusion of Σ_1 into $\mathtt{wsig}(\phi, em)$, that is, $\mathtt{wmon}(\phi, em)(x) = x$ for all symbols x of the signature Σ_1.

— $\mathtt{wmor}(\phi, em): \mathtt{wsig}(\phi, em) \to \Sigma'_2$ coincides with ϕ on Σ_1 and is the identity on the symbols from $\Sigma'_2 - \Sigma_2$

The verification that such $(\mathtt{wsig}, \mathtt{wmon}, \mathtt{wmor})$ actually constitutes a local backward extension, though bowring, is quite straightforward.
Then consider the following very simple case, where there signature have only sorts and all arrows are plain embeddings,

$$\begin{tikzcd}
\{s\} \arrow[r] \arrow[d] & \{s\} \arrow[d] \\
\{s,x\} \arrow[r] & \{s,x\}
\end{tikzcd}$$

But the result is not the same that we get if the identity on $\{s\}$ is factorized through the embedding $\phi_1\colon \{s\} \to \{s,x\}$ and the signature morphism $\phi_2\colon \{s,x\} \to \{s\}$, associating both s and x with s, as follows

$$\begin{tikzcd}
\{s\} \arrow[r, "\phi_1"] \arrow[d] & \{s,x\} \arrow[r, "\phi_2"] \arrow[d] & \{s\} \arrow[d] \\
\{s\} \arrow[r, "\overline{\phi_1}"'] & \{s,x\} \arrow[r, "\overline{\phi_2}"'] & \{s,x\}
\end{tikzcd}$$

where $\overline{\phi_1}$ is the embedding and $\overline{\phi_2}(s) = \overline{\phi_2}(x) = s$. □

Thus, in order to get an overall functor *HMod*, more conditions have to be required from the backward extensions.

Def. 2.12. Let **HMon** be a class of admissible morphisms for an institution $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$. A *backward extension* on **HMon** for $\mathcal{I}$ is a local backward extension `sig`, `mor` and `mon` on **HMon** for $\mathcal{I}$ satisfying the following extra conditions:

4 The choice of `sig`, `mor` and `mon` is natural w.r.t. the first argument:

(a) $\mathtt{sig}(\phi_2 \cdot \phi_1, m) = \mathtt{sig}(\phi_1, \mathtt{mon}(\phi_2, m))$;

(b) $\mathtt{mor}(\phi_2 \cdot \phi_1, m) = \mathtt{mor}(\phi_2, m) \cdot \mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m))$;

(c) $\mathtt{mon}(\phi_2 \cdot \phi_1, m) = \mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m))$.

$$
\begin{tikzcd}
\Sigma_1 \arrow[rr, "\phi_2 \cdot \phi_1"] \arrow[dr, "\phi_1"] \arrow[dd, "\begin{array}{c}\mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m)) = \\ \mathtt{mon}(\phi_2 \cdot \phi_1, m)\end{array}"'] & & \Sigma_3 \arrow[dd, "m"] \\
& \Sigma_2 \arrow[ur, "\phi_2"] \arrow[dd, "\mathtt{mon}(\phi_2, m)", near start] & \\
\begin{array}{c}\mathtt{sig}(\phi_1, \mathtt{mon}(\phi_2, m)) = \\ \mathtt{sig}(\phi_2 \cdot \phi_1, m)\end{array} \arrow[rr, "\mathtt{mor}(\phi_2 \cdot \phi_1, m)", near end] \arrow[dr, "\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m))"'] & & \Sigma_3' \\
& \mathtt{sig}(\phi_2, m) \arrow[ur, "\mathtt{mor}(\phi_2, m)"'] &
\end{tikzcd}
$$

5 The identity as first argument is preserved:

(a) $\mathtt{sig}(Id_{\delta_0(m)}, m) = \delta_1(m)$;

(b) $\mathtt{mor}(Id_{\delta_0(m)}, m) = Id_{\delta_1(m)}$;

(c) $\mathtt{mon}(Id_{\delta_0(m)}, m) = m$. □

It is interesting to note that some apparently good candidates for backward extensions do not satisfy the uniformity conditions and hence cannot be accepted. For instance if all morphisms are acceptable, then let us consider the *minimal extension* that on ϕ and m yields the identity as $\mathtt{mon}(\phi, m)$ and $m \cdot \phi$ as $\mathtt{mor}(\phi, m)$; then for $\phi = Id_\Sigma$ we have m as $\mathtt{mor}(Id_\Sigma, m)$ and hence condition 5 is not satisfied.

Putting together the definitions of $HMod(\Sigma)$ and $HMod(\phi)$ we finally get a functor from **Sign** into $\mathbf{Cat}^{\mathrm{Op}}$.

Prop. 2.13. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and sig, mon, mor be a backward extension on a class **HMon** of admissible morphisms for $\mathcal{I}$. For every $\phi \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$, let $HMod(\phi) \colon HMod(\Sigma_2) \to HMod(\Sigma_1)$ be defined as in Proposition 2.10. Then $HMod$ is a functor from **Sign** into $\mathbf{Cat}^{\mathrm{Op}}$.

Proof. Because of Proposition 2.4 and 2.10, we have that $HMod$ is well-defined; thus we only have to show that it preserves identities and composition in **Sign**.

Identity: let $\Sigma \in |\mathbf{Sign}|$ be a signature, $HA = \langle A', m' \colon \Sigma \hookrightarrow \Sigma' \rangle$, $HB = \langle A'', m'' \colon \Sigma \hookrightarrow \Sigma'' \rangle$ belong to $HMod(\Sigma)$ and $\langle p, m \rangle \colon HA \to HB$ be a model morphism.

$$
\begin{array}{lcl}
HMod(Id_\Sigma)(\langle A', m' \rangle) & = & \text{by definition} \\
\langle Mod(\mathtt{mor}(Id_\Sigma, m'))(A'), \mathtt{mon}(Id_\Sigma, m') \rangle & = & \text{by condition 5b} \\
\langle Mod(Id_{\Sigma'})(A'), \mathtt{mon}(Id_\Sigma, m') \rangle & = & \text{as } Mod \text{ is a functor} \\
\langle A', \mathtt{mon}(Id_\Sigma, m') \rangle & = & \text{by condition 5c} \\
\langle A', m' \rangle & &
\end{array}
$$

Analogously to the previous point we show that $HMod(Id_\Sigma)(\langle p, m\rangle) = \langle p, m\rangle$. Therefore, $HMod(Id_\Sigma)$ is the identity functor over $HMod(\Sigma)$.

Composition: let $\phi_1: \Sigma_1 \to \Sigma_2$ and $\phi_2: \Sigma_2 \to \Sigma_3$ be signature morphisms, $HA = \langle A'_3, m'_3: \Sigma_3 \hookrightarrow \Sigma'_3\rangle$ and $HB = \langle \widetilde{A'_3}, \widetilde{m'_3}: \Sigma_3 \hookrightarrow \widetilde{\Sigma'_3}\rangle$ belong to $HMod(\Sigma_3)$ and $\langle p, m: \Sigma'_3 \hookrightarrow \widetilde{\Sigma'_3}\rangle: HA \to HB$ be a model morphism.

on objects:

$$
\begin{aligned}
&(HMod(\phi_1) \cdot HMod(\phi_2))(HA) &=\\
&HMod(\phi_1)(\langle Mod(\mathtt{mor}(\phi_2, m'_3))(A'_3), \mathtt{mon}(\phi_2, m'_3)\rangle) &=\\
&\langle Mod(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)))(Mod(\mathtt{mor}(\phi_2, m'_3))(A'_3)), \mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m'_3))\rangle &=\\
&\qquad \text{since } Mod \text{ is a functor}\\
&\langle Mod(\mathtt{mor}(\phi_2, m'_3) \cdot \mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)))(A'_3), \mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m'_3))\rangle &=\\
&\qquad \text{by condition 4b}\\
&\langle Mod(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3))(A'_3), \mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m'_3))\rangle &=\\
&\qquad \text{by condition 4c}\\
&\langle Mod(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3))(A'_3), \mathtt{mon}(\phi_2 \cdot \phi_1, m'_3)\rangle &=\\
&HMod(\phi_2 \cdot \phi_1)(\langle A'_3, m'_3\rangle)
\end{aligned}
$$

on morphisms: By definition
$HMod(\phi_2 \cdot \phi_1)(\langle p, m\rangle) = \langle Mod(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3))(p), \mathtt{mon}(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3), m)\rangle$ and
$HMod(\phi_1) \cdot HMod(\phi_2)(\langle p, m\rangle) =$
$HMod(\phi_1)(\langle Mod(\mathtt{mor}(\phi_2, m'_3))(p), \mathtt{mon}(\mathtt{mor}(\phi_2, m'_3), m)\rangle)$ that is equal to the pair with components $Mod(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)))(Mod(\mathtt{mor}(\phi_2, m'_3))(p))$ and
$\mathtt{mon}(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)), \mathtt{mon}(\mathtt{mor}(\phi_2, m'_3), m))$.
Let us consider the first component.

$$
\begin{aligned}
&Mod(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)))(Mod(\mathtt{mor}(\phi_2, m'_3))(p)) &&= \text{ as } Mod \text{ is a functor}\\
&Mod(\mathtt{mor}(\phi_2, m'_3) \cdot \mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)))(p) &&= \text{ by condition 4b}\\
&Mod(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3))(p)
\end{aligned}
$$

Let us consider the second component.

$$
\begin{aligned}
&\mathtt{mon}(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)), \mathtt{mon}(\mathtt{mor}(\phi_2, m'_3), m)) &&= \text{ by condition 4c}\\
&\mathtt{mon}(\mathtt{mor}(\phi_2, m'_3) \cdot \mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)), m) &&= \text{ by condition 4b} \quad \Box\\
&\mathtt{mon}(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3), m)
\end{aligned}
$$

It is worth noting that different backward extensions can be compatible with the same family of admissible morphisms. Let us see a(n artificial but) simple case.

Example 2.14. Let **Sign** be a category with objects $\{X, Y, Z\}$ and as non-trivial arrows only $f: X \to Y$, $g: Y \to Z$ and their composition $h: X \to Z$.
As admissible monomorphisms we consider the identities, g and h; since g and h are not composable, this class is closed under composition.
The definition of the backward extensions of identities is fixed by property 5; thus we have the following diagram, where Σ can be X, Y or Z and m can be Id_Σ or h if $\Sigma = X$, or g if $\Sigma = Y$.

$$\begin{tikzcd}
\Sigma \arrow[r, "Id_\Sigma"] \arrow[d, "\mathtt{mon}(Id_\Sigma, m) = m"'] & \Sigma \arrow[d, "m"] \\
\mathtt{sig}(Id_\Sigma, m) = \Sigma' \arrow[r, "\mathtt{mor}(Id_\Sigma, m) = Id_{\Sigma'}"'] & \Sigma'
\end{tikzcd}$$

Moreover property 3 requires that backward extensions along identities is the morphism itself; thus we have the following diagram, where we can have

— $\Sigma_1 = X$, $\Sigma_2 = Y$ and $\phi = f$, or
— $\Sigma_1 = Y$, $\Sigma_2 = Z$ and $\phi = g$, or
— $\Sigma_1 = X$, $\Sigma_2 = Z$ and $\phi = h$.

$$\begin{tikzcd}
\Sigma_1 \arrow[r, "\phi"] \arrow[d, "\mathtt{mon}(\phi, Id_{\Sigma_2}) = Id_{\Sigma_1}"'] & \Sigma_2 \arrow[d, "Id_{\Sigma_2}"] \\
\mathtt{sig}(\phi, Id_{\Sigma_2}) = \Sigma_1 \arrow[r, "\mathtt{mor}(\phi, Id_{\Sigma_2}) = \phi"'] & \Sigma_2
\end{tikzcd}$$

Therefore, the only relevant backward extension is $\mathtt{sig}(f, g)$, $\mathtt{mor}(f, g)$, $\mathtt{mon}(f, g)$, that we can define in two different ways

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\mathtt{mon}(f,g) = h"'] & Y \arrow[d, "g"] \\
\mathtt{sig}(f,g) = Z \arrow[r, "\mathtt{mor}(f,g) = Id_Z"'] & Z
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "\mathtt{mon}(f,g) = Id_X"'] & Y \arrow[d, "g"] \\
\mathtt{sig}(f,g) = X \arrow[r, "\mathtt{mor}(f,g) = h"'] & Z
\end{tikzcd}$$

It is straightforward (although boring) to verify that the properties of Definition 2.12 are satisfied. □

Let us finally define an institution with the same signatures and sentences as $\mathcal{I}$, but with very abstract models.

Prop. 2.15. Let $\mathcal{I} = (\mathbf{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ be an institution, **HMon** be a family of admissible morphisms, and $\mathtt{sig}$, $\mathtt{mon}$ and $\mathtt{mor}$ be a backward extension on **HMon** for $\mathcal{I}$. Then

$$\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}) = (\mathbf{Sign}, \mathit{Sen}, \mathit{HMod}, \models^H)$$

is an institution, where *HMod* is defined as in Proposition 2.13 and $\models^H$ is defined by:

$$\langle A, m \rangle \models^H_\Sigma \xi \iff A \models_{\Sigma'} \mathit{Sen}(m)(\xi)$$

for each model $\langle A, m \colon \Sigma \hookrightarrow \Sigma' \rangle$ in $|\mathit{HMod}(\Sigma)|$ and each ξ in $\mathit{Sen}(\Sigma)$.

Proof. Because of Proposition 2.13, *HMod* is a functor from **Sign** into $\mathbf{Cat}^{\mathrm{Op}}$ and hence we only have to show that the satisfaction condition holds.
Let $\phi: \Sigma_1 \to \Sigma_2$ be a signature morphism, $\langle A, m: \Sigma_2 \hookrightarrow \Sigma'_2\rangle$ belong to $|HMod(\Sigma_2)|$, and $\xi \in Sen(\Sigma_1)$ be a sentence. Then, by definition of $\models^H$, $\langle A, m\rangle \models^H_{\Sigma_2} Sen(\phi)(\xi)$ iff $A \models_{\Sigma'_2} Sen(m)(Sen(\phi)(\xi))$, i.e. iff $A \models_{\Sigma'_2} Sen(m \cdot \phi)(\xi)$.
Since $\mathcal{I}$ is an institution, $A \models_{\Sigma'_2} Sen(m \cdot \phi)(\xi)$ iff $Mod(m \cdot \phi)(A) \models_{\Sigma_1} \xi$, i.e. iff $Mod(\mathtt{mor}(\phi, m) \cdot \mathtt{mon}(\phi, m))(A) \models_{\Sigma_1} \xi$, because of condition 1, and this is, by definition of $\models^H$, equivalent to $\langle Mod(\mathtt{mor}(\phi, m))(A), \mathtt{mon}(\phi, m)\rangle \models^H_{\Sigma_1} \xi$.
Finally, by definition of $HMod(\phi)$,
$\langle Mod(\mathtt{mor}(\phi, m))(A), \mathtt{mon}(\phi, m)\rangle = HMod(\phi)(\langle A, m\rangle)$. □

The categories $HMod(\Sigma)$ and $Mod(\Sigma)$ are nicely related by the adjoint functors *Emb* and *Flat* described in Definition 2.6, that actually are the components of natural transformations between *HMod* and *Mod* as we will show in the next section. However, some interesting properties do not hold for *HMod*, even if *Mod* satisfies them. For instance, the finite cocompletness of *Mod* does not imply that of *HMod* and hence $\mathcal{H}(\mathcal{I})$ can be non-abstract algebraic (see e.g. (Tarlecki, 1985; Tarlecki, 1986)) while $\mathcal{I}$ is so. Indeed, roughly speaking, the category $HMod(\Sigma)$ is the (disjoint) union of all categories $Mod(\Sigma')$ for some Σ' generalizing Σ, i.e. s.t. there is an admissible monomorphism from Σ into Σ'. Thus, let us consider the case of first-order logic; then, the initial signature Σ_0 is empty (no sorts, nor functions nor predicates), then $HMod(\Sigma_0)$ contains $Mod(\Sigma)$ for all signature Σ and hence is too large to be the terminal object in **Cat**, that is a singleton trivial category.

Since *HMod* is not, in most cases, finitely cocomplete, institutions with *HMod* as model functor cannot be used to define the semantics of specification languages by means of limits and colimits as, for instance, in (R.M. Burstall and J. A. Goguen, 1980); but they are perfectly suitable for those specification languages whose semantics is defined using the notion of validity (for basic specifications) and set theoretic constructions, like in (Sannella and Tarlecki, 1988).

The institution of the hyper-loose (many-sorted first-order with equality) specifications, introduced in (Pepper, 1991), is the very abstract institution over $\mathcal{FOE}$, with admissible morphisms and backward extensions as introduced in the following Application 2.16.

Application 2.16. Let us see that the local backward extension for first-order logic presented in Application 2.9 is also a backward extension.

Proof of 4) Let $\phi_1: \Sigma_1 \to \Sigma_2$ and $\phi_2: \Sigma_2 \to \Sigma_3$, where $\Sigma_1 = (S_1, OP_1, PR_1)$, $\Sigma_2 = (S_2, OP_2, PR_2)$ and $\Sigma_3 = (S_3, OP_3, PR_3)$, be morphisms in **FOESign** and $\iota: \Sigma_3 \hookrightarrow \Sigma'_3$, where $\Sigma'_3 = (S'_3, OP'_3, PR'_3)$ be an admissible morphism in **YMon**, that is a plain embedding.

Moreover let us denote by $\Sigma'_2 = (S'_2, OP'_2, PR'_2)$ the signature $\mathtt{ysig}(\phi_2, \iota)$.

4a) We have to show that the signatures $(S, OP, PR) = \mathtt{ysig}(\phi_1 \cdot \phi_2, \iota)$ and $(\overline{S}, \overline{OP}, \overline{PR}) = \mathtt{ysig}(\phi_1, \mathtt{ymon}(\phi_2, \iota))$ are equal.

Let us see that the two signatures have the same sorts, using the following notation: $n_1 = |S_1|$, $n_2 = |S_2|$, $n_3 = |S_3|$ and $n'_3 = |S'_3|$.

It suffices to show that $|S| = |\overline{S}|$. By definition $|S| = n_1 + n'_3 - n_3$ and $|\overline{S}| = n_1 + n'_2 - n_2$, where n'_2 is the cardinality of the set of sorts of $\texttt{ysig}(\phi_2, \iota)$, that is $n'_2 = n_2 + n'_3 - n_3$. Therefore, $|\overline{S}| = n_1 + n_2 + n'_3 - n_3 - n_2 = |S|$.

Analogously it is possible to show that the two signatures have the same operations and predicates.

4b) Using the same notation as in the previous item, we have to show that $\texttt{ymor}(\phi_2 \cdot \phi_1, \iota) = \texttt{ymor}(\phi_2, \iota) \cdot \texttt{ymor}(\phi_1, \texttt{ymon}(\phi_2, \iota))$.

Let us see that the equality holds for the sort component and leave the analogous proofs for operations and predicates to the reader.

Let us assume $s \in S$. Then, using the notation of the previous point, we have $s = \mathsf{srt}(i)$ for some $1 \leq i \leq n_1 + n'_3 - n_3$ and, by definition

$$\texttt{ymor}(\phi_2 \cdot \phi_1, \iota)(s) = \begin{cases} \phi_2 \cdot \phi_1(s) & \text{if } i \leq n_1 \\ \mathsf{srt}(i - n_1 + n_3) & \text{if } n_1 < i \end{cases}$$

and analogously

$$\texttt{ymor}(\phi_1, \texttt{ymon}(\phi_2, \iota))(s) = \begin{cases} \phi_1(s) & \text{if } i \leq n_1 \\ \mathsf{srt}(i - n_1 + n_2) & \text{if } n_1 < i \end{cases}$$

so that

$$\texttt{ymor}(\phi_2, \iota) \cdot \texttt{ymor}(\phi_1, \texttt{ymon}(\phi_2, \iota))(s) = \begin{cases} \phi_2(\phi_1(s)) & \text{if } i \leq n_1 \\ \mathsf{srt}((i - n_1 + n_2) - n_2 + n_3) & \text{if } n_1 < i \end{cases}$$

as $n_1 < i \leq n_1 + n'_3 - n_3$ implies $n_2 < i - n_1 + n_2$.

4c) Trivial, as there is at the most one injection between signatures.

Proof of 5) Straightforward.

Therefore `ysig`, `ymor` and `ymon` are a backward extension, so that we can use them to define an institution,

$$\mathcal{Y} = \mathsf{ABSTRACT}(\mathcal{FOE}, \mathbf{YMon}, \texttt{ysig}, \texttt{ymor}, \texttt{ymon}),$$

called the institution of *hyper-loose specifications*.

Remark. Since the definition of admissible morphisms and associated backward extensions only depends on the signature category, the choice of **HMon**, `sig`, `mon` and `mor`, can be shared by institutions with the same signatures, disregarding the models and the sentences. In particular, the choice for `ysig`, `ymon` and `ymor` backward extension on the embeddings as admissible morphisms, presented in Application 2.16 for the case of many-sorted signatures (with predicates), applies, hence, in most significant institutions and in particular to the institutions of (conditional) equational specifications of many-sorted total (partial, non-strict) algebras.

Prop. 2.17. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and $\mathcal{I}' = (\mathbf{Sign}, Sen', Mod', \models')$ be institutions and **HMon** be a family of admissible morphisms for $\mathcal{I}$. Then **HMon** is a family of admissible morphisms for $\mathcal{I}'$, too, and any backward extension on **HMon** for $\mathcal{I}$ is a backward extension on **HMon** for $\mathcal{I}'$ as well.

Proof. Trivial. □

Thus, as $\mathsf{ABSTRACT}$ does not affect the signature category, admissible monomorphisms and backward extensions for an institution $\mathcal{I}$ are also such for any institution $\mathsf{ABSTRACT}(\mathcal{I},\ldots)$. Thus, several applications of $\mathsf{ABSTRACT}$ can be performed sequentially, given the ingredients only for the starting institution. In particular we can always consider $\mathcal{I}' = \mathsf{ABSTRACT}(\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon}), \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$, that is, in general, different from $\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$ not only from a technical viewpoint, because the models of $\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$ are pairs $\langle A, m\rangle$, while the object of $\mathcal{I}'$ have the form $\langle\langle A, m\rangle, e\rangle$, but also from an intuitive viewpoint. Indeed, if we regard the symbols in $\Sigma' - \Sigma$ as private for a model (module) $\langle A, m\colon \Sigma \hookrightarrow \Sigma'\rangle$, then a model $\langle\langle A, m\colon \Sigma' \hookrightarrow \Sigma''\rangle, e\colon \Sigma \hookrightarrow \Sigma'\rangle$ has an intermediate level of privacy between the global symbols in Σ and the local symbols in $\Sigma'' - \Sigma'$.

It is also worth noting that, given an admissible monomorphism family $\mathbf{HMon}$ for an institution $\mathcal{I} = (\mathbf{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ and a subcategory $\mathbf{Sign}'$ of $\mathbf{Sign}$, with embedding E, the restriction of $\mathbf{HMon}$ to the elements of $\mathbf{Sign}'$ is a family of admissible monomorphisms for the institutions $\mathcal{I}' = (\mathbf{Sign}', \mathit{Sen} \cdot E, \mathit{Mod} \cdot E, \models)$. Moreover, the restriction of any backward extension $\mathtt{sig}$, $\mathtt{mor}$ and $\mathtt{mon}$ for $\mathcal{I}$ s.t. the extensions of signature morphisms in $\mathbf{Sign}'$ along admissible monomorphisms in $\mathbf{Sign}'$ yield morphisms in $\mathbf{Sign}'$ too, gives a backward extension for $\mathcal{I}'$.

Since abstracting models does not affect the consequence relation between set of sentences, any (complete) entailment system (see e.g. (Meseguer, 1989)) for an institution $\mathcal{I}$ gives an entailment system for any institution $\mathsf{ABSTRACT}(\mathcal{I},\ldots)$ as well.

Def. 2.18. Let $\mathcal{I} = (\mathbf{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ be an institution and let us use the following symbols $\xi, \xi_i \in \mathit{Sen}(\Sigma)$, $\Gamma, \Gamma' \subseteq \mathit{Sen}(\Sigma)$ and $A \in |\mathit{Mod}(\Sigma)|$.
An *entailment system* $\vdash$ for $\mathcal{I}$ is a family $\{\vdash_\Sigma \subseteq \wp(\mathit{Sen}(\Sigma)) \times \mathit{Sen}(\Sigma)\}_{\Sigma \in |\mathbf{Sign}|}$ satisfying the following conditions:

soundness: if $\Gamma \vdash_\Sigma \xi$ and $A \models_\Sigma \gamma$ for all $\gamma \in \Gamma$, then $A \models_\Sigma \xi$;
reflexivity: $\{\xi\} \vdash_\Sigma \xi$ for each $\xi \in \mathit{Sen}(\Sigma)$;
monotonicity: if $\Gamma \vdash_\Sigma \xi$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \vdash_\Sigma \xi$;
transitivity: if $\Gamma \vdash_\Sigma \xi_i$ for all $i \in I$ and $\Gamma \cup \{\xi_i \mid i \in I\} \vdash_\Sigma \eta$, then $\Gamma \vdash_\Sigma \eta$;
$\vdash$-translation: if $\Gamma \vdash_\Sigma \xi$, then $\mathit{Sen}(\phi)(\Gamma) \vdash_{\Sigma'} \mathit{Sen}(\phi)(\xi)$ for any $\phi\colon \Sigma \to \Sigma'$ in $\mathbf{Sign}$.

An entailment is said *complete* iff the following condition holds

completeness: if $A \models_\Sigma \xi$ for all A s.t. ($A \models_\Sigma \gamma$ for all $\gamma \in \Gamma$), then $\Gamma \vdash_\Sigma \xi$.

Moreover, ξ is a *semantic consequence of* Γ, denoted by $\Gamma \models_\Sigma \xi$, iff ($A \models_\Sigma \gamma$ for all $\gamma \in \Gamma$ implies $A \models_\Sigma \xi$) for each $A \in |\mathit{Mod}(\Sigma)|$.

Prop. 2.19. Let $\mathcal{I} = (\mathbf{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ be an institution, $\vdash$ be a (complete) entailment system for $\mathcal{I}$, $\mathbf{HMon}$ be a family of admissible morphisms, and $\mathtt{sig}$, $\mathtt{mon}$ and $\mathtt{mor}$ be a backward extension on $\mathbf{HMon}$.
Then $\vdash$ is a (complete) entailment system also for the institution

$$\mathcal{I}' = \mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}).$$

Proof. Since $\mathcal{I}'$ has the same signatures and sentences as $\mathcal{I}$ it it immediate to see that the reflexivity, monotonicity, transitivity and $\vdash$-translation conditions are satisfied.

Then we have to show that $\vdash$ is sound (complete) for $\mathcal{I}'$, that is $\Gamma \vdash_\Sigma \xi$ implies $\Gamma \models^H_\Sigma \xi$ ($\Gamma \models^H_\Sigma \xi$ implies $\Gamma \vdash_\Sigma \xi$), assuming that $\vdash$ is sound (complete) for $\mathcal{I}$, that is $\Gamma \vdash_\Sigma \xi$ implies $\Gamma \models_\Sigma \xi$ ($\Gamma \models_\Sigma \xi$ implies $\Gamma \vdash_\Sigma \xi$).
Thus, in order to show that soundness (as well as completeness) holds, it suffices to show that $\Gamma \models^H_\Sigma \xi$ iff $\Gamma \models_\Sigma \xi$.
Let us assume that $\Gamma \models^H_\Sigma \xi$ and let $A \in |Mod(\Sigma)|$ be such that $A \models_\Sigma \gamma$ for all $\gamma \in \Gamma$. Then, by definition of $\models^H$, $\langle A, Id_\Sigma \rangle \models^H_\Sigma \gamma$ for all $\gamma \in \Gamma$, so that $\langle A, Id_\Sigma \rangle \models^H_\Sigma \xi$, that is $A \models_\Sigma \xi$. Therefore, $\Gamma \models^H_\Sigma \xi$ implies $\Gamma \models_\Sigma \xi$.
Vice versa, let us assume that $\Gamma \models_\Sigma \xi$ and let $\langle A, m\colon \Sigma \hookrightarrow \Sigma' \rangle \in |HMod(\Sigma)|$ be such that $\langle A, m \rangle \models^H_\Sigma \gamma$ for all $\gamma \in \Gamma$. Then, by definition of $\models^H$, $Mod(m)(A) \models_\Sigma \gamma$ for all $\gamma \in \Gamma$, so that $Mod(m)(A) \models_\Sigma \xi$, that is $\langle A, m \rangle \models^H_\Sigma \xi$. Therefore, $\Gamma \models_\Sigma \xi$ implies $\Gamma \models^H_\Sigma \xi$. □

2.4. *Moving between (Very Abstract) Institutions*

As we have seen in Definition 2.6, the categories of models and very abstract models for a given signature are related each other by a pair of functors. Now we will show that such relation smoothly generalizes to natural transformations between the model and the very abstract model functors. Moreover, we will see that, since *Emb* and *Flat* preserve and reflect validity, they can be used to relate an institution and any very abstract institution built on the top of it by both institution morphisms and maps of institutions.

Thus, each institution morphism and each map of institutions between two underlying institutions can be lifted to work on the very abstract institutions built on them.

Prop. 2.20. Under the hypothesis and using the notation of Proposition 2.13, both $Emb\colon Mod \Rightarrow HMod$ and $Flat\colon HMod \Rightarrow Mod$, defined for each $\Sigma \in |\mathbf{Sign}|$ as in Definition 2.6, are natural transformations.

Proof. By Proposition 2.7, for each signature $\Sigma \in |\mathbf{Sign}|$ both *Emb* and *Flat* are functors. Thus, we only have to show that the naturality diagram commutes for them. Let $\phi\colon \Sigma \to \Sigma'$ be a signature morphism in **Sign** and consider the following diagram.

$$\begin{tikzcd} Mod(\Sigma) \arrow[r, "Emb_\Sigma"] & HMod(\Sigma) \\ Mod(\Sigma') \arrow[u, "Mod(\phi)"] \arrow[r, "Emb_{\Sigma'}"'] & HMod(\Sigma') \arrow[u, "HMod(\phi)"'] \end{tikzcd}$$

Let A belong to $Mod(\Sigma')$; then, by definition of *Emb*, $HMod(\phi)(Emb_{\Sigma'}(A)) = HMod(\phi)(\langle A, Id_{\Sigma'} \rangle)$ and, by definition of *HMod*, $HMod(\phi)(\langle A, Id_{\Sigma'} \rangle)$ is equal to $\langle Mod(\mathtt{mor}(\phi, Id_{\Sigma'}))(A), \mathtt{mon}(\phi, Id_{\Sigma'}) \rangle$. But, by condition 3 of Definition 2.8, $\langle Mod(\mathtt{mor}(\phi, Id_{\Sigma'}))(A), \mathtt{mon}(\phi, Id_{\Sigma'}) \rangle = \langle Mod(\phi)(A), Id_\Sigma \rangle$, that is $Emb_\Sigma \cdot Mod(\phi)(A)$, by definition of *Emb*.
A similar proof applies to morphism translation. Indeed, let $p\colon A \to B$ be a morphism in

$Mod(\Sigma')$; then by definition of Emb, $HMod$ and by condition 3 of Definition 2.8:

$$\begin{array}{ll}
HMod(\phi) \cdot Emb_{\Sigma'}(p) & = \\
HMod(\phi)(\langle p, Id_{\Sigma'}\rangle) & = \\
\langle Mod(\mathtt{mor}(\phi, Id_{\Sigma'}))(p), \mathtt{mon}(\mathtt{mor}(\phi, Id_{\Sigma'}), Id_{\Sigma'})\rangle & = \\
\langle Mod(\phi)(p), \mathtt{mon}(\phi, Id_{\Sigma'})\rangle & = \\
\langle Mod(\phi)(p), Id_{\Sigma}\rangle & = \\
Emb_{\Sigma} \cdot Mod(\phi)(p) &
\end{array}$$

and hence Emb is natural.
Analogously we proceed to show that $Flat$ is natural too. Let $\phi\colon \Sigma \to \Sigma'$ be a signature morphism in **Sign** and consider the following diagram.

$$\begin{tikzcd}
Mod(\Sigma) & HMod(\Sigma) \arrow[l, "Flat_{\Sigma}"'] \\
Mod(\Sigma') \arrow[u, "Mod(\phi)"] & HMod(\Sigma') \arrow[l, "Flat_{\Sigma'}"] \arrow[u, "HMod(\phi)"']
\end{tikzcd}$$

Let $\langle p, m\rangle$ be a homomorphism in $HMod(\Sigma')$ from $\langle A_1, m_1\colon \Sigma' \to \Sigma'_1\rangle$ into $\langle A_2, m_2\colon \Sigma' \to \Sigma'_2\rangle$.
Then $Mod(\phi) \cdot Flat_{\Sigma'}(\langle p, m\rangle)$ is

$$Mod(\phi)(Mod(m_1)(p))\colon Mod(\phi)(Mod(m_1)(A_1)) \to Mod(\phi)(Mod(m_2)(A_2)),$$

i.e. $Mod(m_1 \cdot \phi)(p)\colon Mod(m_1 \cdot \phi)(A_1) \to Mod(m_2 \cdot \phi)(A_2)$.
By definition of $HMod$, $Flat_{\Sigma} \cdot HMod(\phi)(\langle p, m\rangle)$ is the image along $Flat_{\Sigma}$ of the morphism $\langle Mod(\mathtt{mor}(\phi, m_1))(p), \mathtt{mon}(\mathtt{mor}(\phi, m_1), m)\rangle$ from $\langle Mod(\mathtt{mor}(\phi, m_1))(A_1), \mathtt{mon}(\phi, m_1)\rangle$ into $\langle Mod(\mathtt{mor}(\phi, m_2))(A_2), \mathtt{mon}(\phi, m_2)\rangle$, that is $Mod(\mathtt{mon}(\phi, m_1))(Mod(\mathtt{mor}(\phi, m_1))(p))$ from $Mod(\mathtt{mon}(\phi, m_1))(Mod(\mathtt{mor}(\phi, m_1))(A_1))$ into $Mod(\mathtt{mon}(\phi, m_2))(Mod(\mathtt{mor}(\phi, m_2))(A_2))$.
This, by condition 1, becomes $Mod(m_1 \cdot \phi)(p)\colon Mod(m_1 \cdot \phi)(A_1) \to Mod(m_2 \cdot \phi)(A_2)$ and hence $Flat$ is natural. □

It is also worth noting that the functors Emb and $Flat$ preserve and reflect validity. Therefore, $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$ and $(Id_{\mathbf{Sign}}, Id_{Sen}, Flat)$ are both maps of institutions and institution morphisms.

Def. 2.21. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', Sen', Mod', \models')$ be institutions.
A *plain map of institution* (see e.g. (Meseguer, 1989)) $(\Phi, \alpha, \beta)\colon \mathcal{I} \to \mathcal{I}'$ consists of

— a functor $\Phi\colon \mathbf{Sign} \to \mathbf{Sign}'$;
— a natural transformation $\alpha\colon Sen \to Sen' \cdot \Phi$ and
— a natural transformation $\beta\colon Mod' \cdot \Phi \to Mod$

such that for each $\Sigma \in |\mathbf{Sign}|$, each $\xi \in Sen(\Sigma)$ and each $M' \in Mod'(\Phi(\Sigma))$ the following property is satisfied:

$$M' \models'_{\Phi(\Sigma)} \alpha_{\Sigma}(\xi) \text{ iff } \beta_{\Sigma}(M') \models_{\Sigma} \xi.$$

A *morphism of institution* (see e.g. (Burstall and Goguen, 1984; Burstall and Goguen, 1992)) $(\Phi, \alpha, \beta): \mathcal{I} \to \mathcal{I}'$ consists of

— a functor $\Phi: \mathbf{Sign} \to \mathbf{Sign}'$;
— a natural transformation $\alpha: Sen' \cdot \Phi \to Sen$ and
— a natural transformation $\beta: Mod \to Mod' \cdot \Phi$

such that for each $\Sigma \in |\mathbf{Sign}|$, each $\xi' \in Sen' \cdot \Phi(\Sigma)$ and each $M \in Mod(\Sigma)$ the following property is satisfied:

$$M \models_\Sigma \alpha_\Sigma(\xi') \text{ iff } \beta_\Sigma(M) \models'_{\Phi(\Sigma)} \xi'. \square$$

Prop. 2.22. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{HMon}$ be a family of admissible morphisms, and $\mathtt{sig}$, $\mathtt{mon}$ and $\mathtt{mor}$ be a backward extension on $\mathbf{HMon}$; moreover let

$$\mathcal{H} = \mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}) = (\mathbf{Sign}, Sen, HMod, \models^H)$$

be defined as in Proposition 2.15.
Then $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$ is a plain map of institutions from $\mathcal{H}$ into $\mathcal{I}$ and a morphism of institutions from $\mathcal{I}$ into $\mathcal{H}$ and $(Id_{\mathbf{Sign}}, Id_{Sen}, Flat)$ is a plain map of institutions from $\mathcal{I}$ into $\mathcal{H}$ and a morphism of institutions from $\mathcal{H}$ into $\mathcal{I}$.

Proof. To prove that $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$ is a plain map of institutions from $\mathcal{H}$ into $\mathcal{I}$ and a morphism of institutions from $\mathcal{I}$ into $\mathcal{H}$, by definition, it suffices to show that for each $\Sigma \in |\mathbf{Sign}|$, each $\xi \in Sen(\Sigma)$ and each $M \in Mod(\Sigma)$ we have that $M \models_\Sigma \xi$ iff $Emb_\Sigma(M) \models^H_\Sigma \xi$. But this trivially follows from the definition of $\models^H$, because $Emb_\Sigma(M) \models^H_\Sigma \xi$ iff $Mod(Id_\Sigma)(M) \models_\Sigma \xi$, $Emb_\Sigma(M)$ being $\langle M, Id_\Sigma \rangle$.
Analogously, to prove that $(Id_{\mathbf{Sign}}, Id_{Sen}, Flat)$ is a plain map of institutions from $\mathcal{I}$ into $\mathcal{H}$ and a morphism of institutions from $\mathcal{H}$ into $\mathcal{I}$, it suffices to show that for each $\Sigma \in |\mathbf{Sign}|$, each $\xi \in Sen(\Sigma)$ and each $\langle M, m \rangle \in Mod(\Sigma)$ we have that $\langle M, m \rangle \models^H_\Sigma \xi$ iff $Flat_\Sigma(\langle M, m \rangle) \models_\Sigma \xi$. By definition of $\models^H$, $\langle M, m \rangle \models^H_\Sigma \xi$ iff $Mod(m)(M) \models_\Sigma \xi$ i.e., by definition of $Flat$, iff $Flat_\Sigma(\langle M, m \rangle) \models_\Sigma \xi$. $\square$

Since any institution is related to its very abstract generalization by plain maps of institutions (institution morphisms), any map of institutions (institution morphism) generalizes to a map (morphism) between the very abstract generalization of its domain and codomain.

Indeed, let us for instance consider the case of maps of institutions. Let $(\Phi, \alpha, \beta): \mathcal{I} \to \mathcal{I}'$ be a map of institutions, $\mathcal{H}$ be $\mathsf{ABSTRACT}(\mathcal{I}, \ldots)$ and $\mathcal{H}'$ be $\mathsf{ABSTRACT}(\mathcal{I}', \ldots)$, then the composition of $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$, (Φ, α, β) and $(Id_{\mathbf{Sign}'}, Id_{Sen'}, Flat')$ in diagrammatic order is a map of institutions, because it is the composition of maps.

However, it is interesting to note that such composition yields (Φ, α, β'), where β' is defined by $\beta'_\Sigma(\langle M', m' \rangle) = Emb_\Sigma(\beta_\Sigma(M'_{|m'})) = \langle \beta_\Sigma(M'_{|m'}), Id_\Sigma \rangle$. Thus, in the particular case of the identity map of institutions, we have that the above composition gives $(Id_{\mathbf{Sign}}, Id_{Sen}, \overline{\beta})$, where $\overline{\beta}(\langle M, m \rangle) = \langle M_{|m}, Id_\Sigma \rangle$. That is, the identity yields a non-identical map of institutions, associating each pair with the *minimal* element among those pairs having the same projection on the fixed part. Indeed, denoting $M_{|m}$ by A, the pair $\langle Id_A, \widetilde{m} \rangle$ is a morphism from $\langle A, Id_\Sigma \rangle$ into any $\langle \widetilde{M}, \widetilde{m} \rangle$ s.t. $\widetilde{M}_{|\widetilde{m}} = A$.

The existence of *Emb* and *Flat* relates the existence of initial models in the categories of standard and very abstract models. Indeed, since *Emb* and *Flat* preserve and reflect validity, they are also adjoint functors between the standard and the very abstract models of each specification; therefore if the standard models of a specification have an initial object, then its translation along *Emb* is initial for the very abstract models.

Prop. 2.23. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, **HMon** be a family of admissible morphisms, and `sig`, `mon` and `mor` be a backward extension on **HMon**; moreover let

$$\mathcal{H} = \mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}) = (\mathbf{Sign}, Sen, HMod, \models^{H})$$

be defined as in Proposition 2.15 and *Emb*, *Flat* be defined as in Definition 2.6.
Then *Emb* reflects and preserves the initial model of a specification (if any); moreover *Flat* preserves the initial model of a specification.

Proof. Let $T = (\Sigma, \Gamma)$ be a specification for $\mathcal{I}$ and $\mathcal{H}$; let us denote by Mod the model class of T in $\mathcal{I}$, i.e.

$$\mathsf{Mod} = \{A \mid A \in |Mod(\Sigma)| \ s.t. \ A \models_{\Sigma} \gamma \text{ for all } \gamma \in \Gamma\}$$

and by HMod the model class of T in $\mathcal{H}$, i.e.

$$\mathsf{HMod} = \{HA \mid HA \in |HMod(\Sigma)| \ s.t. \ HA \models^{H}_{\Sigma} \gamma \text{ for all } \gamma \in \Gamma\}.$$

Since *Emb* and *Flat* preserve and reflect validity, by Proposition 2.22, they are adjoint functors between Mod and HMod, too, by Proposition 2.7.
Therefore, as left adjoints preserve colimits, *Emb* preserves initiality, i.e. if A is initial in Mod, then $Emb(A)$ is initial in HMod.

Let us assume that $HA = \langle A, m_A \colon \Sigma \hookrightarrow \Sigma' \rangle$ is initial in HMod and show that $Flat(HA)$ is initial in Mod.
Let us first prove that $Emb(Flat(HA))$ is isomorphic to HA and hence it is initial in HMod. Since HA is initial in HMod, there exists a unique $h \colon HA \to Emb(Flat(HA))$. Moreover, by Proposition 2.7, the counit of the adjunction $\epsilon_{HA} = \langle Id_{A_{|m_A}}, m_A \rangle$ is a morphism from $Emb(Flat(HA))$ into HA. Therefore $\epsilon_{HA} \cdot h$ is the unique arrow from HA into itself, i.e. it is the identity.
Then, denoting h by $\langle p, m \colon \Sigma' \hookrightarrow \Sigma \rangle$, $Id_{HA} = \epsilon_{HA} \cdot h = \langle Id_{(A_{|m_A})_{|m}} \cdot p, m_A \cdot m \rangle$ that is $Id_{\Sigma'} = m_A \cdot m$ and $Id_A = Id_{(A_{|m_A})_{|m}} \cdot p$.
Thus, as Mod is a functor, $Id_A = Mod(m)(Id_{Mod(m_A)(A)}) \cdot p = Id_{Mod(m_A \cdot m)(A)} \cdot p = Id_A \cdot p = p$.
Therefore $h \cdot \epsilon_{HA} = \langle Id_A, m \rangle \cdot \epsilon_{HA}$ is $\langle (Id_A)_{|m_A} \cdot Id_{(A_{|m_A})}, m \cdot m_A \rangle = \langle Id_{(A_{|m_A})}, Id_{\Sigma'} \rangle$.
Thus $Emb(Flat(HA))$ is initial in HMod.
Let B belong to Mod; then $Emb(B) \in \mathsf{HMod}$ and hence there is a unique $h \colon Emb(Flat(HA)) \to Emb(B)$, so that its image along *Flat* is a morphism and, since $Flat \cdot Emb$ is the identity, $Flat(h) \colon Flat(HA) \to B$. Moreover if $k \colon Flat(HA) \to B$ is a morphism, then $Emb(k) \colon Emb(Flat(HA)) \to Emb(B)$ and hence $Emb(k) = h$, so that $k = Flat(Emb(k)) = Flat(h)$; therefore $Flat(h)$ is the unique morphism from $Flat(HA)$ into B.
Thus if HA is initial in HMod, then $Flat(HA)$ is initial in Mod and hence *Flat* preserves initiality.

Finally if $Emb(A)$ is initial in HMod, then, as *Flat* preserves initiality, $A = Flat(Emb(A))$ is initial in Mod, i.e. *Emb* reflects initiality. $\square$

3. Enriching sentences by derived ones

3.1. *The operation* EXTEND

While last section was dealing with a generalization of models, that is of the semantic component of a formalism, here an operation is presented, allowing the use of more expressive logic, leaving the language and the semantics unaffected. Particularly interesting results can be obtained by the combination of the two operations, as we will see in the next section.

The starting intuition is that we want to use as sentences on a signature, the sentences built out of the symbols of a richer signature. A well known purely mathematical example is the embedding of first-order logic with equality into standard first-order logic, implementing equality as a special binary predicate, whose interpretation in all models is the identity relation. Thus we can distinguish two steps: the sentences are enriched, by allowing as sentences on a given signature the formulae on that signature enriched by the equality predicate, and then the models are extended in a canonical way to models of the enriched signature, so that the validity of sentences can be borrowed from the standard definition for the extensions.

In the general case the extension operation cannot be performed on some signatures. For instance those signatures where the symbols to be added are already present, with a possibly different semantics, cannot be extended at all. Analogously, some signature morphisms are incompatible with the extensions. Thus, in general, a subcategory of signatures has to be selected as signature category of the result of this operation.

On the semantic side a canonical way of extending the models, in order to define the validity of the new sentences by a standard interpretation of the extra-symbols, is needed. But, since the definition of validity does not involve the categorical structure of the model class, we simply need a function associating each model with its extension, disregarding the model morphisms.

Prop. 3.1. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{Sign}_E$ be a subcategory of $\mathbf{Sign}$ with embedding $E\colon \mathbf{Sign}_E \to \mathbf{Sign}$, EXT be a functor from $\mathbf{Sign}_E$ into $\mathbf{Sign}$ and Ext be a natural transformation from $set \cdot Mod \cdot E$ into $set \cdot Mod \cdot EXT$, where set denotes the functor dropping from a category all non trivial morphisms. Then

$$\mathsf{EXTEND}(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext) = (\mathbf{Sign}_E, Sen_E, Mod_E, \models^E)$$

where $Sen_E = Sen \cdot EXT$, $Mod_E = Mod \cdot E$ and $A \models^E_\Sigma \xi$ iff $Ext(A) \models_{EXT(\Sigma)} \xi$, is an institution.

Proof. The only property that has to be checked is the satisfaction condition.
Let $\phi \in \mathbf{Sign}_E(\Sigma_1, \Sigma_2)$ be a signature morphism, $A \in |Mod_E(\Sigma_2)|$ be a model and $\xi \in Sen_E(\Sigma_1)$ be a sentence.
By definition $A \models^E_{\Sigma_2} Sen_E(\phi)(\xi)$ iff $Ext_{\Sigma_2}(A) \models_{EXT(\Sigma_2)} Sen(EXT(\phi))(\xi)$; since $\mathcal{I}$ is an

institution, by the satisfaction condition for $\mathcal{I}$,

$$Ext_{\Sigma_2}(A) \models_{EXT(\Sigma_2)} Sen(EXT(\phi))(\xi) \quad \text{iff} \quad Mod(EXT(\phi))(Ext_{\Sigma_2}(A)) \models_{EXT(\Sigma_1)} \xi,$$

i.e. since *Ext* is a natural transformation, iff $Ext_{\Sigma_1}(Mod(E(\phi))(A)) \models_{EXT(\Sigma_1)} \xi$, i.e. by definition, iff $Mod_E(\phi)(A) \models^E_{\Sigma_1} \xi$. □

Notice that the EXTEND operation can be used to change the notion of validity. Indeed, if E and EXT are identities, but *Ext* is any natural transformation, then EXTEND produces the same institution, but the definition of validity refers, in the result, to a different model. A sensible application of this construction is, for instance, the definition of an *observational* institution, where each model is associated by *Ext* with its *fully abstract* representation. It is interesting to note that in this way, to prove that the result is actually an institution, it suffices to check that the definition of fully abstraction of a model is compatible with the reducts along signature morphisms.

Let us see a more extensive example of application of the EXTEND operation, that is the use of second-order formulae on first-order models. Indeed, in a first-order model all information needed to describe its second-order extension is present, because the function spaces can be derived from the carriers for the basic values.

Application 3.2. Let us consider the institution of first-order logic. We want to use a richer logic for describing properties of its models, allowing quantification to range not only on values, but also on functions.

The basic idea is to extend a signature with sorts for the function spaces and explicit *apply* operations, whose intended result on a function and arguments for such a function is the result of the function itself.

Accordingly first-order structures are extended, by interpreting the functional sorts by the corresponding function spaces.

For this application, in order to improve readability, it seems more convenient¶ to use *concrete* signatures and to fix a representation for them where names are identifiers, that is strings on a fixed alphabet. Therefore, in the following we assume that, in particular, sorts are strings on a set of symbols including $\rightarrow$.

Let us describe the arguments for EXTEND to get a second-order logic

Extensible signatures. We have to restrict first-order signatures to those where the names we want to use for functional sorts are not already in use. Thus let $\mathbf{FOESign}_{HO}$ be the full subcategory of concrete first-order signatures whose objects are the signatures having sort names that do not contain the symbol $\rightarrow$.

Extensions of signatures. We want to add functional sorts and apply functions. Thus for each $\Sigma = (S, OP, PR)$ let $EXT_{HO}(\Sigma)$ be the signature $(\vec{S}, \vec{OP}, PR)$, where $\vec{S}$ is inductively defined by:

¶ In the example sketched here, the set of sorts of the extended signatures are infinite. Thus, in order to use abstract signatures, we would have to generalize first the category of abstract signatures itself, as suggested in a footnote in Application 2.9. Moreover we should give the renaming mechanism sending each functional sort into its abstract representative. On the bright side, we should not restrict signatures, because we would add new names.

$S \subseteq \vec{S}$ and

if $s_1, \ldots, s_n, s \in \vec{S}$ then $(s_1 \times \ldots \times s_n \to s) \in \vec{S}$;

moreover $\vec{OP}$ consists of all the operations in OP and of the following operations:

$\mathtt{apply} \in \vec{OP}_{\vec{s}\, s_1 \ldots s_n, s}$ for each $\vec{s} = (s_1 \times \ldots \times s_n \to s) \in \vec{S}$ and

$\mathtt{the}op \in \vec{OP}_{\lambda, \vec{s}}$ for each $op \in OP_{s_1 \ldots s_n, s}$ and $n > 0$, where $\vec{s} = (s_1 \times \ldots \times s_n \to s) \in \vec{S}$.

Finally each signature morphism $\phi: \Sigma \to \Sigma'$ is extended to a signature morphism $EXT_{HO}(\phi): EXT_{HO}(\Sigma) \to EXT_{HO}(\Sigma')$ as follows:

— $EXT_{HO}(\phi)$ is inductively defined on $\vec{S}$ by:

$EXT_{HO}(\phi)(s) = \phi(s)$ if $s \in S$ and

$EXT_{HO}(\phi)((s_1 \times \ldots \times s_n \to s)) =$
$(EXT_{HO}(\phi)(s_1) \times \ldots \times EXT_{HO}(\phi)(s_n) \to EXT_{HO}(\phi)(s))$;

— $EXT_{HO}(\phi)$ is ϕ on OP and on PR;

— $EXT_{HO}(\phi)$ is the identity on each $\mathtt{apply}$ operation;

— $EXT_{HO}(\phi)(\mathtt{the}op) = \mathtt{the}\phi(op)$ for each $op \in OP_{s_1 \ldots s_n, s}$.

Extensions of models. Let A be a first-order structure on a signature $\Sigma = (S, OP, PR)$; then $HA = Ext_{HO\,\Sigma}(A)$ is the first-order structure on $EXT_{HO}(\Sigma)$ defined by:

Carriers. They are inductively defined by:

$s^{HA} = s^A$ if $s \in S$ and

$(s_1 \times \ldots \times s_n \to s)^{HA}$ is the set of all functions from $s_1^{HA} \times \ldots \times s_n^{HA}$ into s^{HA} for all $(s_1 \times \ldots \times s_n \to s) \in \vec{S}$;

Operations. There are three cases:

— $op^{HA} = op^A$ for each $op \in OP$;

— $\mathtt{the}op^{HA} = op^A$ for each $op \in OP$;

— $\mathtt{apply}^{HA}(f, a_1, \ldots, a_n)) = f(a_1, \ldots, a_n)$ for each $\mathtt{apply} \in \vec{OP}_{\vec{s}\, s_1 \ldots s_n, s}$, with $\vec{s} = (s_1 \times \ldots \times s_n \to s)$, and all $f \in \vec{s}^{HA} = [s_1^{HA} \times \ldots \times s_n^{HA} \to s^{HA}]$, $a_i \in s_i^{HA}$.

Predicates. $p^{HA} = p^A$ for each $p \in PR$.

It is immediate to check that EXT_{HO} is a functor and that Ext_{HO} is a natural transformation. Therefore $\mathcal{HO} = \mathsf{EXTEND}(\mathcal{FOE}, \mathbf{FOESign}_{HO}, EXT_{HO}, Ext_{HO})$ is an institution.

An example of specification in $\mathcal{HO}$ taking advantage of the higher-order features is the specification of standard models of natural numbers.

spec *Standard* =
sorts *bool*, *nat*
opns *True*, *False*: $\to$ *bool*
Zero: $\to$ *nat*
S: *nat* $\to$ *nat*

axioms

$\forall x, y : nat.S(x) = S(y) \supset x = y$
$True \neq False$
$\forall b : bool.b = True \vee b = False$
$\forall P : nat \to bool.(P(Zero) \wedge (\forall x : nat.P(x) \supset P(S(x)))) \supset \forall x : nat.P(x)$

In each model of *Standard*, the carrier of *bool* has exactly two elements, thus the variables of sort $nat \to bool$ are interpreted in the set of all predicates on the carrier of sort *nat*. Therefore, the last axiom has the effect of restricting the models to those satisfying the induction principle.

Many other slightly different "higher-order" extensions are possible. For instance, we can add only first-order sorts (that are those of the form $s_1 \times \ldots \times s_n \to s_{n+1}$ with all the s_i's in S), getting actually a *second-order* logic (and the example above can still be expressed), or we can interpret non-basic sorts in the extension of the models as the set of all computable functions. However, other apparently good candidates cannot be expressed through the EXTEND operation. For instance, we cannot interpret non-basic sorts in the extension of the models as the set of all expressible functions in some functional language (e.g. some typed λ-calculus), nor, in particular, as the set of the interpretations of the function symbols in the signature, because in that case the extension on models is not natural, because the reduct functor should throw away the elements that are not denoted. □

3.2. *Properties of* EXTEND

The definition of EXTEND in Proposition 3.1 does not assume any relation between a signature and its extension, nor between a model and its extensions. In particular a signature is not required to be a subsignature of its extension, though this property is satisfied in most examples.

As a consequence of this generality, it is not possible to prove that the starting institution and its extension are related by morphisms, nor by maps.

But, under the more restrictive, but very reasonable, hypothesis that a signature is a subsignature of its extension and, accordingly, that the restriction of a model extension to that subsignature gives back the starting model, it is possible to show that the extension of an institution is embedded by an institution morphism into that institution.

Prop. 3.3. Under the hypothesis and using the notation of Proposition 3.1, let η be a natural transformation from E into EXT s.t. each η_Σ is a signature morphism from $E(\Sigma)$ into $EXT(\Sigma)$ in **Sign** and $Ext_\Sigma(M)_{|\eta_\Sigma} = M$ for all models M.
Then $(E, Sen(\eta), Id_{Mod_E})$ is an institution morphism from $\mathsf{EXTEND}(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext)$ into $\mathcal{I}$.

Proof. Since η is a natural transformation from E into EXT, by composing it with the functor Sen, we get a natural transformation from $Sen \cdot E$ into $Sen_E = Sen \cdot EXT$. Thus, the only property that has to be checked is the satisfaction condition. Let $A \in |Mod_E(\Sigma)|$ be a model and $\xi \in Sen(E(\Sigma))$ be a sentence.
By definition $A \models^E_\Sigma Sen(\eta)(\xi)$ iff $Ext_\Sigma(A) \models_{EXT(\Sigma)} Sen(\eta)(\xi)$; since $\mathcal{I}$ is an institution, by

the satisfaction condition for $\mathcal{I}$, $Ext_\Sigma(A) \models_{EXT(\Sigma)} Sen(\eta)(\xi)$ iff $Mod(\eta)(Ext_\Sigma(A)) \models_{E(\Sigma)} \xi$, i.e., since $Ext_\Sigma(M)_{|\eta_\Sigma} = M$ for all models M, iff $A \models_{E(\Sigma)} \xi$. $\square$

A simple example of application of this construction is the introduction of equality in (fragments of either partial or total) first-order logic. In that case, if working with concrete signatures, then the extensible ones are those where the selected equality symbol does not appear as binary predicate, and each such signature is extended by adding the equality as a binary predicate for each sort; otherwise, using abstract signatures, all signatures are extensible and each one is extended by adding for each sort a binary homogeneous predicate (determined by the first free index for that arity).

The models are extended by interpreting the new predicate as identity (in the partial case this corresponds to having *existential* equality) and the η signature morphisms are simply the embeddings.

The rational behind describing institutions by means of operations on them is making the definition of several frameworks tailored for particular applications modularly based on a few, well-understood basic institutions. Thus, it is common to have several different operations, and correspondingly their arguments, for the same institution. For instance, for the institution of first-order logic we have described the extension of sentences allowing higher-order quantification and the extension adding equalities.

An interesting point is, therefore, whether we can compose different extensions. Indeed, let us assume that we have the basic ingredients for two extensions on the same institution. Then it is immediate to see that, in order to sequentially perform both extensions, we only need that one of them code signatures into signatures that can be extended by the other one.

Prop. 3.4. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{Sign}_E$ and $\mathbf{Sign}'_E$ be subcategories of $\mathbf{Sign}$ with embedding E and E', respectively. Moreover, let EXT and EXT' be functors into $\mathbf{Sign}$ respectively from $\mathbf{Sign}_E$ and $\mathbf{Sign}'_E$. Finally, let Ext be a natural transformation from $set \cdot Mod \cdot E$ into $set \cdot Mod \cdot EXT$ and Ext' be a natural transformation from $set \cdot Mod \cdot E'$ into $set \cdot Mod \cdot EXT'$.
If the image of EXT is contained into $\mathbf{Sign}'_E$, then $\mathcal{I}$, $\mathbf{Sign}_E$, $EXT' \cdot EXT$ and $Ext'_{EXT} \circ Ext$ are correct arguments for the operation EXTEND.

Proof. Since the image of EXT is contained into $\mathbf{Sign}'_E$, then $EXT' \cdot EXT$ is well defined. Thus, we only have to prove that $Ext'_{EXT} \circ Ext$ is a natural transformation from $set \cdot Mod \cdot E$ into $set \cdot Mod \cdot EXT' \cdot EXT$.
Let $\phi \colon \Sigma \to \Sigma'$ be a signature morphism in $\mathbf{Sign}_E$; then $EXT(\phi) \colon EXT(\Sigma) \to EXT(\Sigma')$ is a signature morphism in $\mathbf{Sign}'_E$, because the image of EXT is included into $\mathbf{Sign}'_E$.
Therefore, by gluing together the naturality diagram of Ext for ϕ and of Ext' for $EXT(\phi)$, we get the naturality diagram of $Ext'_{EXT} \circ Ext$ for ϕ, as follows:

$$
\begin{tikzcd}[column sep=large, row sep=huge]
|Mod \cdot E(\Sigma)| \arrow[r, "Ext_\Sigma"] & \begin{array}{c}|Mod \cdot EXT(\Sigma)| = \\ |Mod \cdot E' \cdot EXT(\Sigma)|\end{array} \arrow[r, "Ext'_{EXT(\Sigma)}"] & |Mod \cdot EXT' \cdot EXT(\Sigma)| \\
|Mod \cdot E(\Sigma')| \arrow[r, "Ext_{\Sigma'}"'] \arrow[u, "|Mod \cdot E(\phi)|"] & \begin{array}{c}|Mod \cdot EXT(\Sigma')| = \\ |Mod \cdot E' \cdot EXT(\Sigma')|\end{array} \arrow[r, "Ext'_{EXT(\Sigma')}"'] \arrow[u, "\begin{array}{c}|Mod \cdot EXT(\Sigma')| = \\ |Mod \cdot E' \cdot EXT(\phi)|\end{array}"] & |Mod \cdot EXT' \cdot EXT(\Sigma')| \arrow[u, "|Mod \cdot EXT' \cdot EXT(\phi)|"]
\end{tikzcd}
$$

Notice that $E'(\Sigma) = \Sigma$ for each signature Σ in $\mathbf{Sign}_E$ and hence $|Mod \cdot EXT(\Sigma)| = |Mod \cdot E' \cdot EXT(\Sigma)|$. $\square$

In most cases, even if the image of EXT is not included in $\mathbf{Sign}'_E$, it is possible to restrict the extensible signatures to a subcategory $\mathbf{Sign}''_E$ of $\mathbf{Sign}_E$ s.t. its image along EXT is contained in $\mathbf{Sign}'_E$, without loosing too much interesting languages.

For instance, we can sequentially perform the extension adding equalities and that adding functional sorts, in any order, restricting ourselves to those signatures where the equality symbol is not used as a binary predicate and, at the same time, sorts are not built using the $\to$ symbol. The construction yields the same result for both composition orders, as it is always the case for extensions simply adding new symbols, because union is commutative.

4. Combining EXTEND and ABSTRACT

4.1. *Properties of combinations of* EXTEND *and* ABSTRACT

Combining the operations proposed in the previous sections we can modularly produce quite powerful frameworks.

As in the case of composition of extensions, also for the combination of extension and model abstraction, it is often the case that we have the basic ingredients for the operations on the same basic institution and that we would like to perform them sequentially.

Thus, let us assume that we are able to apply EXTEND to some tuple $(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext)$ and ABSTRACT to some tuple $(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$, for the same institution $\mathcal{I}$. The compatibility conditions between $(\mathbf{Sign}_E, EXT, Ext)$ and $(\mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$ to be required in order to be able to perform both operations are very simple and intuitive. Indeed, if we want first to extend sentences, since the signatures are in this way restricted to $\mathbf{Sign}_E$, we only need that `sig`, `mor` and `mon` can be restricted as well, that is, that on extensible morphisms they yield extensible morphisms, too. On the other hand, if we want first to generalize models, then we need to be able to lift Ext to work on abstract models. At this aim, it is sufficient that EXT preserves admissible monomorphisms, to be able to translate the models, and commutes with `sig`, `mor` and `mon`, to get a natural transformation.

Prop. 4.1. Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{HMon}$ be a family of admissible morphisms, and `sig`, `mon` and `mor` be a backward extension on $\mathbf{HMon}$. Moreover, let $\mathbf{Sign}_E$ be a subcategory of $\mathbf{Sign}$ with embedding E, let EXT be a functor from $\mathbf{Sign}_E$ into $\mathbf{Sign}$ and let Ext be a natural transformation from $set \cdot Mod \cdot E$ into $set \cdot Mod \cdot EXT$.

1 The family of all admissible morphisms belonging to $\mathbf{Sign}_E$, denoted by $\mathbf{HMon}_E$, is a family of admissible morphisms for $\mathbf{Sign}_E$.
Moreover, if $\mathtt{sig}(\phi, m)$, $\mathtt{mor}(\phi, m)$ and $\mathtt{mon}(\phi, m)$ belong to $\mathbf{Sign}_E$ for all admissible monomorphisms m and signature morphisms ϕ in $\mathbf{Sign}_E$, then $\mathtt{sig}_E$, $\mathtt{mon}_E$ and $\mathtt{mor}_E$, defined by $\mathtt{sig}_E(\phi, m) = \mathtt{sig}(\phi, m)$, $\mathtt{mon}_E(\phi, m) = \mathtt{mon}(\phi, m)$ and $\mathtt{mor}_E(\phi, m) = \mathtt{mor}(\phi, m)$ for all $m \in \mathbf{HMon}_E$ and all ϕ in $\mathbf{Sign}_E$, is a backward extension for $\mathbf{Sign}_E$ and $\mathbf{HMon}_E$.

2 If $\mathbf{HMon}$ is contained into the morphisms of $\mathbf{Sign}_E$ and EXT both preserves admissible monomorphisms and commutes with $\mathtt{sig}$, $\mathtt{mon}$ and $\mathtt{mor}$, that is

— $\mathtt{sig}(EXT(\phi), EXT(m)) = EXT(\mathtt{sig}(\phi, m))$,

— $\mathtt{mor}(EXT(\phi), EXT(m)) = EXT(\mathtt{mor}(\phi, m))$ and

— $\mathtt{mon}(EXT(\phi), EXT(m)) = EXT(\mathtt{mon}(\phi, m))$,

then *HExt*, defined by $HExt_\Sigma(\langle A, m{:}\, \Sigma \hookrightarrow \Sigma'\rangle) = \langle Ext_{\Sigma'}(A), EXT(m)\rangle$ is a natural extension from $set \cdot HMod \cdot E$ into $set \cdot HMod \cdot EXT$.

Moreover if the hypothesis of both points are satisfied, then the applications of EXTEND and $\mathsf{ABSTRACT}$ in the two orders yield the same result, that is

$$\mathsf{ABSTRACT}(\mathsf{EXTEND}(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext), \mathbf{HMon}_E, \mathtt{sig}_E, \mathtt{mon}_E, \mathtt{mor}_E)$$
$$=$$
$$\mathsf{EXTEND}(\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}), \mathbf{Sign}_E, EXT, HExt)$$

Proof.

1 Since $\mathbf{Sign}_E$ is a subcategory of $\mathbf{Sign}$, it is closed under morphism compositions and identities. Therefore, as $\mathbf{HMon}$ is closed under morphism compositions and identities too, so $\mathbf{HMon}_E$ is and hence it is a family of admissible morphisms for $\mathbf{Sign}_E$.
Moreover, if $\mathtt{sig}(\phi, m)$, $\mathtt{mor}(\phi, m)$ and $\mathtt{mon}(\phi, m)$ belong to $\mathbf{Sign}_E$ for all admissible morphisms m and signature morphisms ϕ in $\mathbf{Sign}_E$, then $\mathtt{sig}_E$, $\mathtt{mon}_E$ and $\mathtt{mor}_E$ are well defined and inherit from $\mathtt{sig}$, $\mathtt{mon}$ and $\mathtt{mor}$ the satisfaction of the commutativity properties required for backward extensions.

2 Since EXT preserves admissible morphisms, each $HExt_\Sigma$ is properly defined.
Let us see that *HExt* is natural. Let us consider a signature morphism $\phi{:}\, \Sigma_1 \to \Sigma_2$ in $\mathbf{Sign}_E$ and a model $\langle A, m{:}\, \Sigma_2 \hookrightarrow \Sigma_2'\rangle$ in $HMod(\Sigma_2) = HMod(E(\Sigma_2))$.
Then, by direct application of the definitions, we have:

$HExt_{\Sigma_1}(HMod(\phi)(\langle A, m\rangle)) =$	by definition of $HMod(\phi)$
$HExt_{\Sigma_1}(\langle Mod(\mathtt{mor}(\phi, m))(A), \mathtt{mon}(\phi, m)\rangle) =$	by definition of *HExt*
$\langle Ext_{\mathtt{sig}(\phi,m)} Mod(\mathtt{mor}(\phi, m))(A), EXT(\mathtt{mon}(\phi, m))\rangle =$	by naturality of *Ext*
$\langle Mod(EXT(\mathtt{mor}(\phi, m)))(Ext_{\Sigma_2'}(A)), EXT(\mathtt{mon}(\phi, m))\rangle =$	as EXT commutes w.r.t. $\mathtt{mon}$, $\mathtt{mor}$
$\langle Mod(\mathtt{mor}(EXT(\phi), EXT(m)))(Ext_{\Sigma_2'}(A)), \mathtt{mon}(EXT(\phi), EXT(m))\rangle =$	
$=$	by definition of $HMod(EXT(\phi))$
$HMod(EXT(\phi))(\langle Ext_{\Sigma_2'}(A), EXT(m)\rangle) =$	by definition of *HExt*
$HMod(EXT(\phi))(HExt_{\Sigma_2}(\langle A, m\rangle))$	

Let us assume, now, that both hypothesis are satisfied. Then, by definition, we have:

$$\begin{array}{ll}
\mathsf{ABSTRACT}(\mathsf{EXTEND}(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext), \mathbf{HMon}_E, \mathtt{sig}_E, \mathtt{mon}_E, \mathtt{mor}_E) & = \\
\mathsf{ABSTRACT}((\mathbf{Sign}_E, Sen \cdot EXT, Mod \cdot E, \models^E), \mathbf{HMon}_E, \mathtt{sig}_E, \mathtt{mon}_E, \mathtt{mor}_E) & = \\
(\mathbf{Sign}_E, Sen \cdot EXT, HMod_E, \models') & = \mathcal{I}'
\end{array}$$

where $HMod_E(\Sigma)$ consists of all pairs $\langle A, m\colon \Sigma \hookrightarrow \Sigma'\rangle$ with A an object of $Mod \cdot E(\Sigma') = Mod(\Sigma')$ and m an admissible morphism in $\mathbf{HMon}_E$. But, as $\mathbf{HMon}$ is contained into the signature morphisms in $\mathbf{Sign}_E$, $\mathbf{HMon} = \mathbf{HMon}_E$ and hence $HMod_E(\Sigma) = HMod(\Sigma)$. Therefore, $HMod_E = HMod \cdot E$. Moreover, $\models'$ is defined by:
$\langle A, m\colon \Sigma \hookrightarrow \Sigma'\rangle \models_{\Sigma'} \xi$ iff $Mod(m)(A) \models^E_{\Sigma'} \xi$ iff $Ext_{\Sigma'}(Mod(m)(A)) \models_{EXT(\Sigma')} \xi$.
Analogously,

$$\begin{array}{ll}
\mathsf{EXTEND}(\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mon}, \mathtt{mor}), \mathbf{Sign}_E, EXT, HExt) & = \\
\mathsf{EXTEND}((\mathbf{Sign}, Sen, HMod, \models^H), \mathbf{Sign}_E, EXT, HExt) & = \\
(\mathbf{Sign}_E, Sen \cdot EXT, HMod \cdot E, \models'') & = \mathcal{I}''
\end{array}$$

Moreover, $\models''$ is defined by:
$\langle A, m\colon \Sigma \hookrightarrow \Sigma'\rangle \models''_{\Sigma} \xi$ iff $HExt_{\Sigma}(\langle A, m\rangle) \models^H_{EXT(\Sigma)} \xi$ iff
$\langle Ext_{\Sigma}(A), EXT(m)\rangle \models^H_{EXT(\Sigma)} \xi$ iff $Mod(EXT(m))(Ext_{\Sigma}(A)) \models_{EXT(\Sigma')} \xi$
Thus, since $Ext_{\Sigma'}(Mod(m)(A)) = Mod(EXT(m))(Ext_{\Sigma}(A))$ by naturality of Ext, we have that $\models'$ and $\models''$ are equal. Therefore, $\mathcal{I}' = \mathcal{I}''$. $\square$

4.2. *Very abstract first-order specifications*

Let us see, as a motivating example, the application of the sentence extension operation to the institution of very-abstract data type, built generalizing the models of first-order logic.

Here, to get a more intuitive and readable result, we are using an algebraic metalanguage presenting a *concrete* signature, that is intended to represent its abstract counterpart. This corresponds to having a *parser* translating each concrete symbol into an abstract one, for instance associating each new sort with $\mathsf{srt}(i)$ in the order defined by the declaration, starting from the first free index (that is the cardinality of the sort set of the source signature plus 1) and analogously proceeding for operations and predicates.

Application 4.2. The idea is to extend the institution $\mathcal{Y}$, introduced in Application 2.16, with appropriate formulae for expressing requirements on the (extra part of the) signatures of the very abstract models. Obviously, there are different ways to choose these requirements. Here we present a rather general and powerful choice, that we think appropriate for many reasonable applications. However, at the same time, we are introducing only those constructs needed by the specification examples presented here. Our idea is to give the possibility to express both purely syntactic conditions on the extra part of the models (e.g. requiring the (non) existence of an operation or a predicate whose functionality satisfies some conditions) but also semantic one (e.g. requiring the (non) existence of an operation or a predicate whose interpretation satisfies some conditions, as commutativity). The practice of using very abstract specifications of abstract data

types (shortly VAS) will show whether this choice is appropriate, possibly suggesting improvements and modifications.
Then the institution for first-order VAS is built by applying the operation EXTEND to the institution $\mathcal{Y}$ and the underlying idea is to take as new formulae on a signature Σ the formulae of classical first-order logic with equality on the signature enriching Σ by sorts, operations and predicates for handling the syntactic elements on Σ (e.g.: sorts, operations, predicates, variables, terms, formulae, ...) and their interpretations. Let us now list the parameters for EXTEND.

The extendible signatures. We have to restrict the admissible signature morphisms. Indeed, having formulae stating the (non) existence and the (dis)equality of the signature symbols, non-injective or non-surjective signature morphisms may do not preserve or reflect validity. For example, let us consider a non-injective signature morphism $\phi: \Sigma_2 \to \Sigma_1$ between signatures having only sorts and no operations nor predicates, $\Sigma_1 = (\{srt\}, \emptyset, \emptyset)$ and $\Sigma_2 = (\{srt_1, srt_2\}, \emptyset, \emptyset)$, defined in the only possible way: $\phi(srt_1) = srt = \phi(srt_2)$.
Then, let ξ be the formula $\exists\ x, y: sort\ .\ x \neq y$, where the type *sort* is interpreted in all extensions of models as the set of sorts of their signature. Now, ξ is satisfied by any Σ_2-model of the form $\langle A_{|\phi}, Id_{\Sigma_2}\rangle$, that is the translation along ϕ of the Σ_1-model $\langle A, Id_{\Sigma_1}\rangle$, while its translation along ϕ does not hold for $\langle A, Id_{\Sigma_1}\rangle$.
If, vice versa, we consider a non-surjective morphism $\phi': \Sigma_1 \to \Sigma_2$, defined by $\phi'(srt) = srt_1$, then the formula $\forall\ x, y: sort\ .\ x = y$ holds for each such $\langle A_{|\phi}, Id_{\Sigma_1}\rangle$, but its translation along ϕ' does not hold for $\langle A, Id_{\Sigma_2}\rangle$.
Therefore we can only extend signature isomorphisms.

Extending signatures. We want to add to each signature sorts representing the elements of the signature itself and functions and predicates to manipulate them. In order to get terms of such extra sorts as close as possible to their "meta" counterparts, we are using an algebraic specification language with heavy overloading, mixfix notation and silent operations, but it is intended to represent an abstract signature, so that the following axioms are unambiguous. Different specification languages could be adopted in order to get a clearer distinction between the levels or, vice versa, to let the users forget that there are two levels.
Let $SYNT$: **VSign** $\to$ **FOESign** be the functor defined by:

on objects: for each signature $\Sigma = (S, OP, PR)$ in **VSign**, let us define

sig $SYNT(\Sigma) =$
enrich Σ **by**
sorts $sort, sort\_seq, atom, formula, opn, pred, var, var\_seq, term, term\_seq$
opns $s: \to sort$ for all $s \in S$
$f: \to opn$ for all $f \in OP$
$Arity: opn \to sort\_seq$
$Type: opn \to sort$
$p: \to pred$ for all $p \in PR$
$PArity: pred \to sort\_seq$
$\lambda: \to sort\_seq$

$\_\cdot\_: sort \times sort\_seq \to sort\_seq$
$\_(\_): opn \times term\_seq \to term$
$Var_n: \to var$ for all $n \in N$
$\lambda: \to var\_seq$
$\_\cdot\_: var \times var\_seq \to var\_seq$
$\_: var \to term$
$\lambda: \to term\_seq$
$\_,\_: term \times term\_seq \to term\_seq$
$\_(\_): pred \times term\_seq \to atom$
$\_=\_: term \times term \to atom$
$\_: atom \to formula$
$\neg\ \_: formula \to formula$
$\_ \supset \_, \_ \wedge \_, \_ \vee \_, \_ \equiv \_: formula \times formula \to formula$
$\forall\ \_ : \_ . \_, \exists\ \_ : \_ . \_: var\_seq \times sort \times formula \to formula$

preds $Holds: formula$
$HasSort: term \times sort$
$HaveSorts: term\_seq \times sort\_seq$
$Basic: opn$
$Basic\text{-}gen: term$

on morphisms: $SYNT$ is the obvious extension leaving the new symbols unaffected, that is $SYNT(\phi)(\sigma) = \phi(\sigma)$ if $\sigma \in \Sigma$ or σ is a constant of type *sort*, *opn* or *pred*, otherwise $SYNT(\phi)(\sigma) = \sigma$.

It is easy to see that $SYNT$ is a functor.

Extending models. For all $\Sigma = (\overline{S}, \overline{OP}, \overline{PR}) \in |\mathbf{VSign}|$ the function $Synt_\Sigma: |YMod(\Sigma)| \to |YMod(SYNT(\Sigma))|$ is defined interpreting each new symbol as its meta-level counterpart. Thus, for each $\langle A, m: \Sigma \hookrightarrow \Sigma_1\rangle$, where $\Sigma_1 = (S, OP, PR)$, let us denote by $T_{\widetilde{\Sigma}}(\widetilde{X})$ the (standard) term algebra on the following subsignature of $SYNT(\Sigma)$:

sig $\widetilde{\Sigma} =$
sorts $sort, sort\_seq, atom, formula, opn, pred, var, var\_seq, term, term\_seq$
opns $\lambda: \to sort\_seq$
$\_\cdot\_: sort \times sort\_seq \to sort\_seq$
$\_(\_): opn \times term\_seq \to term$
$Var_n: \to var$ for all $n \in N$
$\lambda: \to var\_seq$
$\_\cdot\_: var \times var\_seq \to var\_seq$
$\_: var \to term$
$\lambda: \to term\_seq$
$\_,\_: term \times term\_seq \to term\_seq$
$\_(\_): pred \times term\_seq \to atom$
$\_=\_: term \times term \to atom$
$\_: atom \to formula$
$\neg\ \_: formula \to formula$
$\_ \supset \_, \_ \wedge \_, \_ \vee \_, \_ \equiv \_: formula \times formula \to formula$
$\forall\ \_ : \_ . \_, \exists\ \_ : \_ . \_: var\_seq \times sort \times formula \to formula$

and the family of variables $\widetilde{X}$:

$$\widetilde{X}_{sort} = S \quad \widetilde{X}_{opn} = OP \quad \widetilde{X}_{pred} = PR \quad \widetilde{X}_s = \emptyset \text{ otherwise}$$

Then $Synt_\Sigma(A) = B$, where

$s^B = s^A$ for all $s \in S$
$f^B = f^A$ for all $f \in OP$
$p^B = p^A$ for all $p \in PR$
$s^B = T_{\widetilde{\Sigma}}(\widetilde{X})_s$ otherwise
$Arity^B(f) = w$ for all $f \in OP_{w,s}$
$Type^B(f) = s$ for all $f \in OP_{w,s}$
$PArity^B(p) = w$ for all $p \in PR_w$
$f^B = f^{T_{\widetilde{\Sigma}}(\widetilde{X})}$ otherwise
$Holds^B(\xi)$ iff $\xi \in FOESen(\Sigma_1)$ and $A \models^{FOE}_{\Sigma_1} \xi$
$HasSort^B(t, s)$ iff $t \in T_{\Sigma_1}(T_{\widetilde{\Sigma}}(\widetilde{X})_{var})_s$
$HaveSorts^B$ is inductively defined by:

- $HaveSorts^B(\lambda, w)$ iff $w = \lambda$
- $HaveSorts^B(t, wt, s \cdot ws)$ iff $HaveSorts^B(wt, ws)$ and $HasSort^B(t, s)$

$Basic^B(f)$ iff $f \in \overline{OP}$
$Basic\text{-}gen^B = \cup_{s \in \overline{S}} T_\Sigma(T_{\widetilde{\Sigma}}(\widetilde{X})_{var})_s$

Since the morphisms in **VSign** are isomorphisms, obviously *Synt* is a natural transformation.
Therefore, we can apply the EXTEND operation and get an institution.

$$\mathcal{YFOE} = \mathsf{EXTEND}(\mathcal{Y}, \mathbf{VSign}, SYNT, Synt),$$

where $\mathcal{Y} = \mathsf{ABSTRACT}(\mathcal{FOE}, \mathbf{YMon}, \mathtt{ysig}, \mathtt{ymor}, \mathtt{ymon})$, is the specification of the *first-order very abstract* specifications.

Let us now see some examples of specifications in the institution $\mathcal{YFOE}$, motivating our choice of sentence extension.

Example 4.3. We specify the fundamental requirements on a module for handling labelled transition trees without completely fixing the interface. The designer in charge of realizing such module is allowed to devise a nice choice of extra constructors for trees, but it cannot add operations modifying parts of a tree, so that it is possible to give implementations where repeated common subtrees are shared.

spec LTT =
enrich $LAB, STATE$ **by**
sorts $tree, sons$
– fixed components of the interface
opns $\Lambda: \rightarrow sons$
$\langle \_, \_ \rangle \ \&\ \_ : lab \times tree \times sons \rightarrow sons$
$T: state \times sons \rightarrow tree$
axioms
– properties of the fixed part of the interface

$\langle l,t\rangle \;\&\; \langle l,t\rangle \;\&\; sn = \langle l,t\rangle \;\&\; sn$
$\langle l_1,t_1\rangle \;\&\; \langle l_2,t_2\rangle \;\&\; sn = \langle l_2,t_2\rangle \;\&\; \langle l_1,t_1\rangle \;\&\; sn$

- properties of the variable part of the interface
  - each constructor for *tree* is derived

  $\forall\, op\colon opn\,.\, Type(op) = tree \supset \exists\, t : term\,.\, HasSort(t, tree) \wedge Basic\text{-}gen(t) \wedge$
  $\quad \exists\, Var_1 \ldots Var_n\colon var\,.\, \exists\, s_1 \ldots s_n\colon sort\,.$
  $\quad\quad Holds(\forall\, Var_1\colon s_1 \,.\, \ldots \forall\, Var_n\colon s_1\,.\, op(Var_1,\ldots,Var_n) = t)$
  - each constructor for *sons* is derived

  $\forall\, op\colon opn\,.\, Type(op) = sons \supset \exists\, t : term\,.\, HasSort(t, sons) \wedge Basic\text{-}gen(t) \wedge$
  $\quad \exists\, Var_1 \ldots Var_n\colon var\,.\, \exists\, s_1 \ldots s_n\colon sort\,.$
  $\quad\quad Holds(\forall\, Var_1\colon s_1 \,.\, \ldots \forall\, Var_n\colon s_1\,.\, op(Var_1,\ldots,Var_n) = t)$

For example the operation $1_{Ary}\colon state \times lab \times tree \to tree$ building unary trees and defined by $1_{Ary}(s,l,t) = T(s, \langle l,t\rangle \;\&\; \Lambda)$ can be added to the interface, while the following $Replac(where, l, new)$, substituting *new* for each l-labelled subtree of *where*, cannot:

$$\begin{array}{l}
Replac\colon tree \times lab \times tree \to tree\\
Replac(T(s,sn),l,t) = T(s, Replac'(sn,l,t))\\
Replac'\colon sn \times lab \times tree \to sn\\
Replac'(\Lambda,l,t) = \Lambda\\
Replac'(\langle l,t\rangle \;\&\; sn, l, t') = \langle l,t'\rangle \;\&\; Replac'(sn,l,t')\\
l \neq l' \supset Replac'(\langle l,t\rangle \;\&\; sn, l', t') = \langle l,t\rangle \;\&\; Replac'(sn,l',t') \quad \Box
\end{array}$$

Another application of very-abstract specification, is the description of properties required on the local structure of actual parameters for parameterized specifications.

Example 4.4. In most specification languages constructs are provided to describe parameterized specifications, that are (partial) functions yielding a specification, that is a class of models, for any given value of the parameter(s) specification(s). The type of the expected argument is usually described by a specification, too and an actual parameter is acceptable if it is a subclass of the formal parameter. Thus, the heading of a parameterized specification has form $SP(X\colon VAS) = \ldots$ and an instantiation $SP(SP1)$ is correct if $Mod(SP1) \subseteq Mod(VAS)$.
The specification *VAS* provides the names for the minimal structure required from the parameter, in order to be able to define the body of the parameterized specification.
For instance let us consider the following trivial example of a list specification, parameterized on the type of the list elements. Thus *VAS0* has one sort, *elem*, no operations, no predicates and no axioms, and a specification of list with only constructors is as follows.

spec $LIST(X\colon VAS0) =$
enrich X **by**
sorts *list*
opns $\Lambda\colon \to list$
$\quad\quad$ $\_ :: \_\colon elem \times list \to list$

Now, we intuitively could instantiate *LIST* on a specification of integer numbers in order to get lists of integers, but the names introduced by *LIST*, like *list* and Λ, should be unused in the actual parameter in order to avoid name clashes.
While many specification languages provide means to apply a parameterized specification

to arguments on signatures different from that of the formal parameter, by using so called *fitting morphisms*, the problem of preventing name clashes is usually left to side conditions, but cannot be expressed within the logic of the specification itself.
Instead, using first-order very abstract specifications we have a powerful tool to characterize the admissible actual parameters of parameterized algebraic specifications.
For example, in the case of $LIST$, we can substitute the parameter specification

spec *VAS1* =
enrich *VAS0* **by**
axioms
$\forall\, s\colon sort\,.\, s \neq list$

for *VAS0* and be ensured that no name clashes can happen during correct instantiations. Notice that the symbols Λ and $\_ :: \_$ can be overloaded, but, as *list* does not appear in the parameter signature, they cannot be confused with the same notation introduced by $LIST$.
A more sophisticated example is the following specification, implementing sets using list.

spec $SET\_FROM\_LIST(X\colon VAS2)$ =
enrich X **by**
axioms
$x::x::l = x::l$
$x_1::x_2::l = x_2::x_1::l$

where the parameter specification *VAS2*, to be consistent with the axioms of SET_FROM_LIST, cannot have operations nor predicates distinguishing lists on the basis of their length or of the order of their elements.

spec *VAS2* =
sorts $elem, list$
opns $\Lambda\colon \to list$
$\_ :: \_\colon elem \times list \to list$
preds $equiv\colon list$
axioms
$\forall\, l, l' : list\,.\, equiv(l, l') \;\equiv\; equiv(l', l)$
$\forall\, l : list\,.\, equiv(l, l)$
$\forall\, l : list\,.\,\forall\, e : elem\,.\,\neg\, equiv(e::l, \Lambda)$
$\forall\, l : list\,.\,\forall\, e : elem\,.\, equiv(e::e::l, e::l)$
$\forall\, l : list\,.\,\forall\, x_1, x_2 : elem\,.\, equiv(x_1::x_2::l, x_2::x_1::l)$
$\forall\, w, w'\colon sortseq\,.\,\forall\, f\colon opn\,.\, Arity(f) = w \cdot list \cdot w' \supset$
$\forall\, \overline{t}, \overline{t'} : term\_seq\,.\, HaveSorts(\overline{t}, w) \;\wedge\; HaveSorts(\overline{t'}, w') \supset$
$Holds(\forall\, Var_1, Var_2 : list\,.\, equiv(Var_1, Var_2) \supset (f(\overline{t}, Var_1, \overline{t'}) = f(\overline{t}, Var_2, \overline{t'})))$
$\forall\, w, w'\colon sort\_seq\,.\,\forall\, p\colon pred\,.\, PArity(p) = w \cdot list \cdot w' \supset$
$\forall\, \overline{t}, \overline{t'} : term\_seq\,.\, HaveSorts(\overline{t}, w) \;\wedge\; HaveSorts(\overline{t'}, w') \supset$
$Holds(\forall\, Var_1, Var_2 : list\,.\, equiv(Var_1, Var_2) \supset (p(\overline{t}, Var_1, \overline{t'}) \;\equiv\; p(\overline{t}, Var_2, \overline{t'})))$

The axioms of *VAS2* prevent to instantiate SET_FROM_LIST with actual parameters which can cause inconsistencies, as a specification of lists with predicates checking if a list has length n (a natural number).

Let us finally see another application of very-abstract specifications, where it is useful, not to say necessary, to be able to pick up an operation, possibly in the non-fixed part of a specification, for each sort. In order to achieve this result, we combine very-abstract specifications with the higher-order extension. Thus, we get a carrier $sort \Rightarrow opn$ and any constant in it represents a parameterized family of operations in the signature.

Application 4.5. The motivating example of the construction we are going to introduce is the *specification of the abstract actor structures*.
In her recent work on formal modelization of actor systems, C. Talcott has presented a class of structures, called *Abstract Actor Structures*, see (Talcott, 1996), by giving their relevant properties. Such structures can be characterized as the models of a first-order very abstract specification (with term-generation constraints), but not of a first-order (with term-generation constraints) specification.
The relevant point is that the actors in one system use values which can be completely different by those used in another system and such values can be built also using the "actor names". Furthermore, on any sort of such structures a renaming operation is defined, which given a bijective mapping ρ over the actor names renames each occurrence of an actor name in an element of that sort accordingly with ρ. For example, if the values are lists of integers and of actor names, then the application of such renaming on a bijective mapping ρ and the following list $1\ n_1\ 2\ n_2\ 0$, where n_1 and n_2 are actor names, will produce $1\ \rho(n_1)\ 2\ \rho(n_2)\ 0$.
Thus we want to be able to express the following points:

— for each sort s we have a function $Rename_s$;
— the typing of each such $Rename_s$ is $mapping \times s \to s$
— the semantics of each such $Rename_s$ is (mutually) inductively defined by a set of rules, one for each operation $f\colon s_1 \times \ldots \times s_n \to s$ with $s \neq name$, of the form

$$Rename_s(\rho, f(t_1, \ldots, t_n)) = f(Rename_{s_1}(\rho, t_1), \ldots, Rename_{s_n}(\rho, t_n)).$$

Moreover, we have the inductive basis $Rename_{name}(\rho, n) = \rho(n)$.

By a very abstract specification we can express all the above requirements, but the first. In order to be able to denote a sort indexed family of functions we combine the very abstract specifications with the sentence extension introducing higher-order sorts.
Let us consider the institution

$$\mathcal{VFOEHO} = \mathsf{EXTEND}(\mathcal{VFOE}, \mathbf{FOESign}_{HO}, EXT_{HO}, Ext_{HO})$$

where, with a slight notational abuse, we denote by $\mathbf{FOESign}_{HO}$ its restriction to $\mathbf{VSign}$. Since the sets of extra symbols introduced by the two sentence extensions are disjoint, we can sequentially perform both applications starting from the signatures that can be extended in both senses (see e.g. Proposition 3.4).
Now, without fixing the values used by the actors, nor the signatures used to manipulate them, we can qualify "Abstract Actor Structures" by means of a specification in $\mathcal{VFOEHO}$.
Let us see the more significant part of such specification.

spec AAS =
– fixed part
sorts $name, state, actor, message, value, mapping, \ldots$

opns $(\_)\_: name \times state \to actor$ **generator of** $actor$
$\_ \lhd \_: name \times value \to message$ **generator of** $message$
$\_(\_): mapping \times name \to name$
$Rename: sort \to opn$
$Rename^*: mapping \times term\_seq \to term\_seq$
...
axioms
$\forall\, s: sort \,.\, Arity(Rename(s)) = mapping, s$
$\forall\, s: sort \,.\, Type(Rename(s)) = s$
$Holds(\forall\, Var_1: mapping \,.\, \forall\, Var_2: name \,.\, Rename(name)(Var_1, Var_2) = Var_1(Var_2))$
$\forall\, \rho: mapping \,.\, Rename^*(\rho, \lambda) = \lambda$
$\forall\, s: sort \,.\, \forall\, \rho: mapping \,.\, \forall\, t: term \,.\, \forall\, wt: term\_seq \,.$
$HasSort(t, s) \supset Rename^*(\rho, t, wt) = Rename(s)(\rho, t), Rename^*(\rho, wt)$
$\forall\, op: opn \,.\, \forall\, \bar{t}: term\_seq \,.\, HaveSorts(\bar{t}, Arity(op)) \supset$
$Holds(\forall\, Var_1: mapping \,.\, Rename(Type(op))(Var_1, op(\bar{t})) = op(Rename^*(Var_1, \bar{t}))$
...

4.3. *Very Abstract Entity Specifications*

Entity algebras, where *entity* stands for processes, either simple or structured (i.e. several processes interacting together), see (Reggio, 1991), provide a formal framework for the process specification. Each entity has associated an *identity* in such a way that it is possible to retrieve its entity subcomponents depending on their identities and its concurrent/distributed structure. Moreover, such structure can be graphically represented in a way that makes the subcomponent relationships and the subcomponent sharing explicit. This latter feature is very roughly based on the idea that "the concurrent/distributed structure of processes in an entity algebra is given by the algebraic structure of such algebra" (also supported by J. Meseguer (Meseguer, 1992)). Thus, a specification expressing abstract requirements on the concurrent/distributed structure of some processes modelled by entity algebras will naturally be a very abstract specification, i.e. having models with different signatures, i.e. modelling processes with different concurrent/distributed structures.

Technically, we have the basic institution of first-order entity specifications

$$\mathcal{E} = (\mathbf{ESign}, ESen, EMod, \models^E)$$

where

— **ESign** is a category whose objects (*entity signature*) are pairs $E\Sigma = (\Sigma, E)$, where $\Sigma = (S, OP, PR)$ is a many-sorted first-order signature (an object of **FOESign**) and $E \subseteq S$ such that for each $s \in E\Sigma$ there exist:
  - some sorts $ent(s)$, $id(s)$, $lab(s) \in S$ (entities of type s, their identities and the labels of their transitions respectively, s is the sort of their bodies);
  - an operation $\_ : \_ : id(s) \times s \to ent(s) \in OP$ (entity constructor, which taken a body and an identity returns an entity) and
  - a predicate $\_ \xrightarrow{\_} \_ : ent(s) \times lab(s) \times ent(s) \in PR$ (describing the activity of the entities by means of labelled transitions);

and whose morphisms are those of **FOESign** preserving entity sorts and the related auxiliary sorts, operations and predicates.

— *ESen* is the restriction of *FOESen* to **ESign**.

— $EMod(E\Sigma)$ is the subcategory of $FOEMod(E\Sigma)$, whose objects are the entity algebras and $EMod(\phi: E\Sigma_1 \to E\Sigma_2) = FOEMod(\phi: E\Sigma_1 \to E\Sigma_2)$, defined in (Reggio, 1991)[||].

— $EA \models^E \xi \Leftrightarrow EA \models^{FOE} \xi$.

The conditional specifications in $\mathcal{E}$ under reasonable conditions admit initial models, and so they can be used to specify the design of some particular concurrent system, because in such cases the structure of the system is fully determined.

For requirement specifications, instead, we use the very abstract specifications of the institution given, by using the two operations ABSTRACT and EXTEND, as follows.

First, we define

$$\mathcal{YE} = \mathsf{ABSTRACT}(\mathcal{E}, \mathbf{EMon}, \mathtt{esig}, \mathtt{emor}, \mathtt{emor})$$

where **EMon** includes the elements of **YMon** which are also morphisms in **ESign**; `esig`, `emor`, `emor` are the restrictions of `ysig`, `ymon`, `ymor` to **ESign** and **EMon** (in (Reggio, 1991) it is shown that such restrictions are well-defined, i.e. they return signatures and morphisms in **ESign**); then

$$\mathcal{VE} = \mathsf{EXTEND}(\mathcal{YE}, \mathbf{VESign}, COMPS, Comps)$$

where **VESign** is the subcategory of **ESign** s.t. it has the same objects and whose only morphisms are the isomorphisms. *COMPS* adds to an entity signature some predicates for testing which are the subcomponents of the entities (as *Is_Sub_Entity* in the following example) and *Comps* is defined accordingly.

Example 4.6. We specify the class of all structured processes where deadlocks never happen without making assumptions on their concurrent structure (i.e. without "over specification") by a very abstract entity specification.

spec *NO_DEADLOCKS* =
– basic signature
esorts *system* – we have at least entities of sort *system*
axioms
– if a system cannot perform any activity, then
$\forall\, es, l\,.\ \nexists\, es', l\,.\, es \xrightarrow{l} es' \supset$
– each of its subcomponents cannot perform any activity
$\forall\, ec\,.\,(ec\ Is\_Sub\_Entity\ es \supset \nexists\, ec', l'\,.\, ec \xrightarrow{l'} ec')$

(recall that e_1 *Is_Sub_Entity* e_2 holds whenever e_1 is a subcomponent of e_2). □

[||] The precise definition of *entity algebras* is too complex to be accommodate within an example. But intuitively an algebra represents a distributed structure, and hence is an acceptable entity algebra, iff identities are unique within each entity, are preserved by the transitions and each entity is structured, that is if the corresponding carriers are basically term-generated.

5. Conclusions and further work

We have presented two operations on institutions, ABSTRACT and EXTEND, allowing the modular constructions of institutions for very abstract specifications and studied their properties.

To be able to use in practice very abstract first-order specifications, we still need to develop an appropriate specification language for making them more readable and making simpler to write them.

It is interesting to note that the use of EXTEND is not limited to build very abstract institutions, but can be used also in other case, see e.g. in Section 3 for building institution for observational specifications and for second-order logic. Thus, operations on institution could be also a nice tools to simplify the work of checking that a formal framework is an institution.

From the practice of formal methods for software specification, it is easy to intuit that other operations are needed in order to get a sufficiently powerful language for the compositional definition of meta-formalisms. Some more operations are presented in (Cerioli and Reggio, 1993), but these are case studies rather than an organic presentation of the reasonable set of operations.

We think that a careful analysis of more case studies is still needed, in order to get an intuition of the basic constructs constituting the wanted metalanguage for assembling formalisms. Some initial work (see (Cerioli and Reggio, 1995)) shows that a large number of institution used in specification method for concurrent systems may be built by starting from some basic institutions with a set of basic operations roughly corresponding to restricting the signatures, restricting the models and extending the sentences (this one corresponds to use EXTEND).

Furthermore, after having determined an appropriate set of operations, we will have to study their properties, mainly w.r.t. “moving among institutions” and about their compositions so to be able to decide whether two expressions built by such operations denote the same formalism.

Acknowledgements

We are glad to thank the anonymous referee for his/her positive suggestions and criticisms, that helped us (we hope) to improve the paper and encouraged us to find better applications to illustrate our results.

References

Astesiano, E. and Reggio, G. (1993a). A Metalanguage for the Formal Requirement Specification of Reactive Systems. In Woodcock, J. and Larsen, P., editors, *Proc. FME'93: Industrial-Strength Formal Methods*, number 670 in Lecture Notes in Computer Science. Springer Verlag, Berlin.

Astesiano, E. and Reggio, G. (1993b). Specifying Reactive Systems by Abstract Events. In *Proc. of Seventh International Workshop on Software Specification and Design (IWSSD-7)*. IEEE Computer Society, Los Alamitos, CA.

Beierle, C. and Voss, A. (1987). Viewing Implementations as an Institution. In Pitt, D., Poigné, A., and Rydeheard, D., editors, *Proceedings of Category Theory and Computer Science*, number 283 in Lecture Notes in Computer Science, pages 196–218, Berlin. Springer Verlag.

Burstall, R. and Goguen, J. (1984). Introducing Institutions. In Clarke, E. and Kozen, D., editors, *Logics of Programming Workshop*, number 164 in Lecture Notes in Computer Science, pages 221–255. Springer Verlag, Berlin.

Burstall, R. and Goguen, J. (1992). Institutions: Abstract Model Theory for Specification and Programming. *Journal of the Association for Computing Machinery*, 39(1):95–146.

Cerioli, M. and Reggio, G. (1993). Algebraic Oriented Institutions. In Nivat, M., Rattray, C., Rus, T., and Scollo, G., editors, *Algebraic Methodology and Software Technology (AMAST'93)*, Workshops in Computing. Springer Verlag, London.

Cerioli, M. and Reggio, G. (1994). Institutions for Very Abstract Specifications. In Ehrig, H. and Orejas, F., editors, *Recent Trends in Data Type Specification*, number 785 in Lecture Notes in Computer Science, pages 113–127. Springer-Verlag, Berlin.

Cerioli, M. and Reggio, G. (1995). Modular Construction of Formalisms within the SMoLCS Method for the Specification of Concurrent Systems. Talk presented at the First FLIRT Workshop, Genova, October 1995.

Meseguer, J. (1989). General Logic. In *Logic Colloquium'87*. North-Holland, Amsterdam.

Meseguer, J. (1992). Conditional Rewriting as a Unified Model of Concurrency. *T.C.S.*, 96:73–155.

Pepper, P. (1991). Transforming Algebraic Specifications – Lessons Learnt from an Example. In Möller, B., editor, *Proc. IFIP TC2 Conf. on Constructing Programs from Specifications, Pacific Grove, California, May 1991*, pages 1–27. Elsevier, Amsterdam.

Reggio, G. (1991). Entities: an Institution for Dynamic Systems. In Ehrig, H., Jantke, K., Orejas, F., and Reichel, H., editors, *Recent Trends in Data Type Specification*, number 534 in Lecture Notes in Computer Science, pages 244–265. Springer Verlag, Berlin.

Reggio, G. (1993). Event Logic for Specifying Abstract Dynamic Data Types. In Bidoit, M. and Choppy, C., editors, *Recent Trends in Data Type Specification*, number 655 in Lecture Notes in Computer Science, pages 292–309. Springer Verlag, Berlin.

Reggio, G., Morgavi, A., and Filippi, V. (1992). Specification of a High-Voltage Substation. Technical Report PDISI-92-12, DISI – Università di Genova, Italy.

Reichwein, G. and Fiadeiro, J. (1992). A Semantic Framework for Interoperability. Technical report, Departemento de Matematica, Instituto Superior técnico, Universidade técnica de Lisboa.

R.M. Burstall and J. A. Goguen (1980). The Semantics of Clear, a Specification Language. In Bjørner, D., editor, *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, number 86 in Lecture Notes in Computer Science, pages 292–332, Berlin. Springer Verlag.

Sannella, D. and Tarlecki, A. (1988). Specifications in an Arbitrary Institution. *Information and Computation*, 76.

Talcott, C. L. (1996). Interaction Semantics for Components of Distributed Systems. In *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'96*.

Tarlecki, A. (1985). On the Existence of Free Models in Abstract Algebraic Institutions. *T.C.S.*, 37(3):269–304.

Tarlecki, A. (1986). Quasi-varieties in Abstract Algebraic Institutions. *Journal of Computer and System Science*, 33:333–360.