Math. Struct. in Comp. Science (1998), *vol.* 8, *pp.* 17–66. Printed in the United Kingdom
© 1998 Cambridge University Press

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 October 1996; revised May 1997

Two operations are presented for a modular approach to the definition of frameworks for rigorous development of software, formally represented as institutions.
The first generalizes models, allowing them to have more structure than the minimum required by their declared signatures, as happens for software modules, which may have local routines that do not appear in their interface.
The second 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 also studied.

1. Introduction

In the last twenty years, several different formal approaches have been proposed supporting rigorous software development. Even restricting attention to *algebraic* frameworks only, the variations are many, depending on the signatures (ranging 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, such as 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and in all development phases, from the requirement to the design, if any, would be a *monster* and would become out of date in a short time.

[†] 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.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

But, on the other hand, having so many possibilities is confusing, especially for naive 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 problems means the required formalism also changes. 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 on, this situation is to have the means for building the formalisms themselves modularly, 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, which has all the features required by the particular problem under examination, but is as simple as possible.

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

The need for such a modular approach to 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, and for an application in the database field, see Reichwein and Fiadeiro (1992). Another example, found in Sannella and Tarlecki (1988), is the *extension by universal closure* of a given institution and a set of its signature morphisms, where sentences are enriched along these signature morphisms, with the extra symbols regarded 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 caters for the fact that models possibly have more structure than the minimum required by their signatures, as happens for software modules, which may have 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 of the possible local functionalities, stating properties not only of the models (*i.e.,* about the interpretations), but also of the syntaxes themselves (*i.e.,* about the actual structure of the models).

Concrete instances of these constructions can be found in the description of institutions for *very abstract specifications* in the field of concurrency, such as, 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) and Reggio (1993), each having several variants, such as with first-order, conditional or equational logic, or with partial, non-strict or generalized models, and so on.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Another application of the very abstract operation, this time 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 of the syntax of the specifications cannot be imposed. Extending the sentences also, we get the institution of very abstract data types, which supports the specification of high-level requirements on both modules and their interfaces (*e.g.*, constraints either on the number of operations or on the number of arguments of the operations, due to limits of the admissible implementations).

Let us now consider the institution of entity algebras (Reggio 1991), which provides a formal framework for algebraic specifications of concurrent systems, where some signature operations are used to describe the concurrent structure explicitly, (*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, but satisfying common properties. Moreover, these examples have already been used in some industrial case studies of the specification, at different levels of abstraction, of a substation for electric power distribution (Reggio *et al.* 1992).

The very abstract operation is described modularly as the composition of two basic operations on institutions: ABSTRACT, which abstracts the models on a signature Σ by regarding as abstract Σ-models the actual models on each 'extension' of Σ, and EXTEND, which 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, such as equality).

The arguments of ABSTRACT are an institution and a family of signature 'extensions', which 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 ABSTRACT itself. Therefore if two possible extensions are available for the same (signature) institution, it is possible to apply both constructions sequentially. As a paramount instance of ABSTRACT application, here we show the construction of signature extensions for 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 may be expected, the result of an application of 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. Conversely, each standard model is obviously also a very abstract model, which happens to have no local structure. This relationship is formalized by saying that institution morphisms and

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 the validity of the new sentences can be easily defined. Therefore, EXTEND affects both sentences and the 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, which 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 simplifies to a 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 (in particular that built starting from many-sorted logic) by the expressive capability needed 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 a 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 with respect to the syntax of another one. 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 with respect to syntax

In mathematical practice it is quite common to regard rich algebraic structures, such as fields or rings, as poorer structures, such as 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, 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 underlying group result in one group and, more interestingly, we cannot, for instance, use the product(s) in order to prove properties of the sum. The second interpretation is that a ring in itself *is* a group,

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

i.e., groups are all those entities that have *at least* the group operations, but can also 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 practice to have (private) local definitions, which are 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 the complexity of algorithms or simplify termination proofs and so on. 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) and 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, which 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, which applied to a logical framework $\mathscr{I}$ yields a framework over $\mathscr{I}$, 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 a signature.

Definition 2.1. An *institution* $\mathscr{I}$ (see *e.g.*, Burstall and Goguen (1984)) consists of a category **Sign** of *signatures*, a functor $Sen : \mathbf{Sign} \to \mathbf{Set}$ giving the set of *sentences* over a signature, a functor $Mod : \mathbf{Sign} \to \mathbf{Cat}^{\mathrm{Op}}$ giving the category of *models* on a signature, and a *satisfaction relation* $\models \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$ for each Σ object in **Sign**, sometimes denoted by $\models_\Sigma$, such that for each morphism $\phi : \Sigma \to \Sigma'$ in **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 following we assume that $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ is an institution and discuss the elements needed in order to build an institution $\mathscr{H} = \mathsf{ABSTRACT}(\mathscr{I}, \ldots)$, with the same syntax as $\mathscr{I}$ (signature and sentences), but whose models are allowed to have some extra structure.

2.1. *Abstract models on* Σ

Intuitively, in $\mathscr{H}$ 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 $\mathscr{H}$ are the Σ'-models in $\mathscr{I}$, 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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

us 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 also result in an extension. Therefore, it is natural to formalize the admissible morphisms as the arrows of a subcategory having the same object class as the signature category.

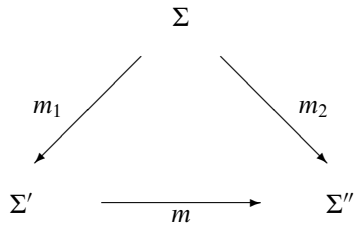
Definition 2.2. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution. Then a family of *admissible morphisms* for $\mathscr{I}$ is any subcategory **HMon** of **Sign** such that $|\mathbf{HMon}| = |\mathbf{Sign}|$.
Here and in the following we write $m: \Sigma \hookrightarrow \Sigma'$ to denote the fact 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 now consider the arrows between these new models in order to get a category. Since abstract models are pairs, 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 also 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 **HMon** such that the following diagram commutes



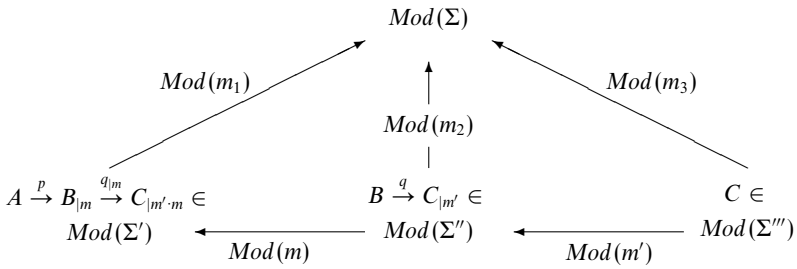
Thus, if such an m exists, B is an algebra on an extension of the actual signature of A, and hence it is natural to choose as first component of the model morphism, a Σ-morphism from A into $B_{|m}$, preserving all the structure of A and not only the minimum required by Σ.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Definition 2.3. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and **HMon** a family of admissible morphisms for $\mathscr{I}$.
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: $HMod(\Sigma)(\langle A, m_1 : \Sigma \hookrightarrow \Sigma'\rangle, \langle B, m_2 : \Sigma \hookrightarrow \Sigma''\rangle) =$
$\{\langle p, m\rangle \mid m \in \mathbf{HMon}(\Sigma', \Sigma''), m \cdot m_1 = m_2 \text{ and } p \in Mod(\Sigma')(A, B_{|m})\}$;
identities: $Id_{\langle A, m:\Sigma \hookrightarrow \Sigma'\rangle} = \langle Id_A, Id_{\Sigma'}\rangle$;
composition: $\langle q, m' : \Sigma'' \hookrightarrow \Sigma'''\rangle \cdot \langle p, m : \Sigma' \hookrightarrow \Sigma''\rangle = \langle q_{|m} \cdot p, m' \cdot m\rangle$.

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



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

Proposition 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 $\langle A, m_1 : \Sigma \hookrightarrow \Sigma'\rangle$ into $\langle B, m_2 : \Sigma \hookrightarrow \Sigma''\rangle$ the Σ-morphisms in $Mod(\Sigma)$ from $A_{|m_1}$ into $B_{|m_2}$. However, in this case 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) and (their renamed structures) are isomorphic, in accordance 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.

Proposition 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. The proof is a straightforward check of the definition. □

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

The models on a signature Σ can be regarded naturally as very abstract models on that signature, seen as an 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*.

Definition 2.6. Using the notation of Definition 2.3, let $Emb : Mod(\Sigma) \to HMod(\Sigma)$ 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 $Flat : HMod(\Sigma) \to Mod(\Sigma)$ 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 $\langle p, m \rangle \in Mod(\Sigma)(\langle A_1, m_1 \rangle, \langle A_2, m_2 \rangle)$, $Flat(\langle p, m \rangle) = p_{|m_1}$.

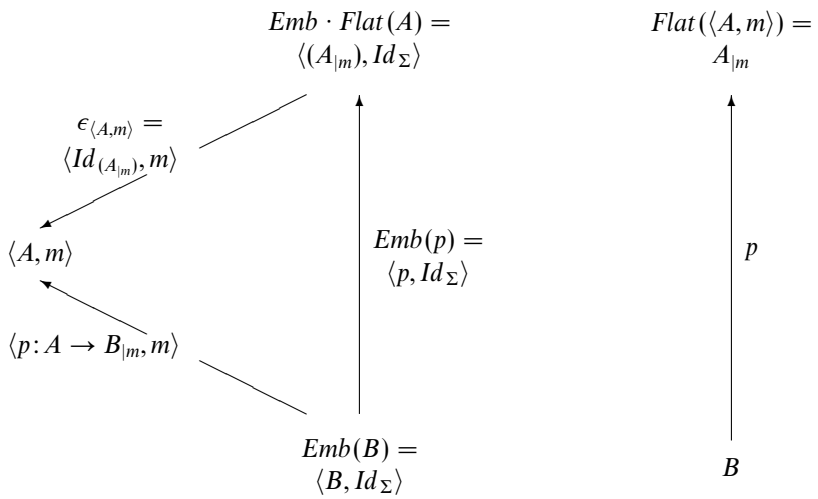
It is straightforward to check that both *Emb* and *Flat* are functors. Moreover, they are adjoint to each other, so *Emb* is a coreflexive subcategory inclusion.

Proposition 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 $\langle p, m' \rangle : Emb(B) \to Emb \cdot Flat(\langle A, m \rangle)$ in $HMod(\Sigma)$ the unique $q : B \to Flat(\langle A, m \rangle)$ such that $\epsilon_{\langle A,m \rangle} \cdot Emb(q) = \langle p, m' \rangle$ is p itself.

Since $\langle p, m' \rangle$ is a morphism in $HMod(\Sigma)$ from $Emb(B) = \langle B, Id_\Sigma \rangle$ into $\langle A, m \rangle$, $m' \cdot Id_\Sigma = m$, that is, $m' = m$, and p is a morphism in $Mod(\Sigma)$ from B into $A_{|m}$, that is $p : B \to Flat(\langle A, m \rangle)$.

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.



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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 us 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, which 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 **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 an extension. Moreover, if ϕ can be extended to a morphism ϕ' between Σ_1' and Σ_2', 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:

$$\begin{array}{ccc} \Sigma_1 & \xrightarrow{\ \phi\ } & \Sigma_2 \\ {\scriptstyle e}\,\vdots & & \downarrow{\scriptstyle m} \\ \Sigma_1' & \xrightarrow[\ \phi'\ ]{} & \Sigma_2' \end{array}$$

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 (though 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'.

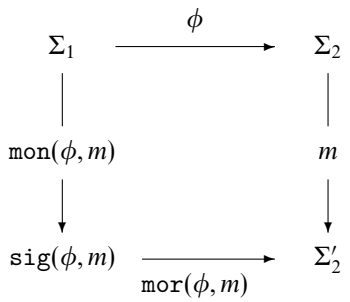
Thus, in the following definition we require that *local backward extensions* are selected satisfying the minimal conditions sufficient for each $HMod(\phi)$, defined by translation of models along such ϕ''s, to be a functor.

Definition 2.8. Let **HMon** be a class of admissible morphisms for an institution $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$. A *local backward extension* on **HMon** for $\mathscr{I}$ 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, that is, $m \cdot \phi = \mathtt{mor}(\phi, m) \cdot \mathtt{mon}(\phi, m)$:

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

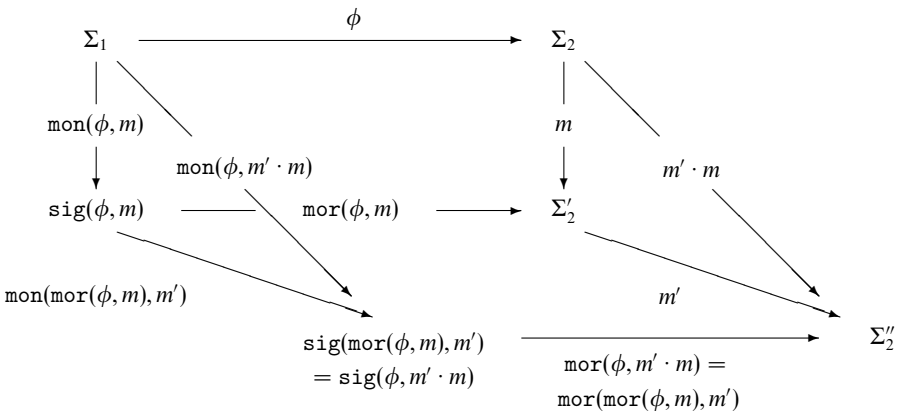


2 The choice of `sig`, `mor` and `mon` is natural with respect to the second argument:

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



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: $\mathtt{mor}(\phi, m) = m \cdot \phi$ (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 make much sense, because the extension is trivial and, as we will see later, 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 satisfy Condition 3c if ϕ is not an identity, because $\mathtt{mon}(\phi, Id_{\delta_1(\phi)}) = \phi$.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

The conditions required from a local backward extension, besides the technical requirements for the proofs of functoriality for $HMod(\phi)$, are determined by the intuition that a local backward extension along $\phi: \Sigma_1 \to \Sigma_2$ and $m: \Sigma_2 \hookrightarrow \Sigma_2'$ 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 $\mathscr{I}$, 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, 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.,*

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

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 $0 :\to nat$
$Reset :\to time$
$\varnothing :\to set$
$S : nat \to nat$
$Ins : nat \times set \to set$
$Clock : time \to time$

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 $Zero :\to int$
$\lambda :\to list$
$Inc, Dec : int \to int$
$Push : int \times list \to list$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

and by the signature morphism ϕ defined by

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

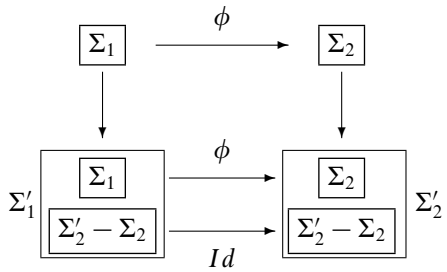
Let us finally consider an extension of Σ_2, where a boolean sort is added:

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

Then the expected extension of Σ_1 is

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

More generally, 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 in order to get an intuition for the possible solutions. In the following, we will use *local* for symbols belonging to an extension but not to the visible part of the signature.

Let us consider the previous example again, 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 $True, False : \rightarrow bool$
$Isempty : list \rightarrow bool$
$Tail : list \rightarrow list$
$Head : list \rightarrow int$

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $True, False :\rightarrow bool$
$Isempty : set \rightarrow bool$
$Tail : set \rightarrow set$
$Head : set \rightarrow nat$
$Head : set \rightarrow time$

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 rather than a set equipped with an arity function.

In accordance with 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**
$IsOrdered : list \rightarrow bool$
$Sort : list \rightarrow list$

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 it 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**
$Sort : set \rightarrow set$

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**
$S : int \rightarrow int$

If we naively apply the technique suggested at the beginning, we have two incarnations of S in the backward extension, one originating from the enrichment by the local structure from Σ_2^{IV} and the other already present in the source. Unfortunately, as for each fixed arity the function symbols form a set, this leads to an undue 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**
$S' : nat \rightarrow nat$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 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 us to introduce new names to avoid undue identifications.

The last case is actually the only problematic one 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 arbitrarily choose a representative. In order to be sure that we are not discarding needed admissible morphisms, we fix many details of the representatives for the 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 the 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 $\mathscr{U}_S$, with enumeration function $\mathsf{srt}: \mathbb{N} \to \mathscr{U}_S$[†], a denumerable universe of operation symbols $\mathscr{U}_O$, with enumeration function $\mathsf{opn}: \mathbb{N} \to \mathscr{U}_O$ and a denumerable universe of predicate symbols $\mathscr{U}_P$, with enumeration function $\mathsf{prd}: \mathbb{N} \to \mathscr{U}_P$. Then an *abstract signature* $\Sigma = (S, OP, PR)$ is a first-order signature such that

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

Let **FOESign** be the category of abstract signatures, and let

$$\mathscr{FOE} = (\mathbf{FOESign}, FOESen, FOEMod, \models^{FOE})$$

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

[†] In order to allow infinite signatures as well, it suffices to use an enumeration $\mathsf{srt}: \mathbb{N} \times \mathbb{N} \to \mathscr{U}_S$ 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 such that $\mathsf{srt}(i, j) \in S$. Analogously for functions and predicates. However, many games on indexes have to be plaied to keep the construction straight.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

interpreted as a presentation in an algebraic language for the following abstract signatures (the 'abstract' names are assigned in declaration order):

sig $\Sigma_1 =$
sorts $\mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3)$
opns $\mathsf{opn}(1) : \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \times \mathsf{srt}(3) \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1) : \mathsf{srt}(2) \rightarrow \mathsf{srt}(2)$

sig $\Sigma_2 =$
sorts $\mathsf{srt}(1), \mathsf{srt}(2)$
opns $\mathsf{opn}(1) : \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1), \mathsf{opn}(2) : \mathsf{srt}(1) \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \times \mathsf{srt}(2) \rightarrow \mathsf{srt}(2)$

sig $\Sigma_2' =$
sorts $\mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3)$
opns $\mathsf{opn}(1) : \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1), \mathsf{opn}(2) : \mathsf{srt}(1) \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \times \mathsf{srt}(2) \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1),\ \mathsf{opn}(2) : \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1) : \mathsf{srt}(2) \rightarrow \mathsf{srt}(3)$

and, accordingly, the morphism† ϕ is

$$\phi(\mathsf{srt}(1)) = \mathsf{srt}(1) \quad \phi_{\lambda,\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \quad \phi_{\mathsf{srt}(1),\mathsf{srt}(1)}(\mathsf{opn}(1)) = \mathsf{opn}(1)$$
$$\phi(\mathsf{srt}(2)) = \mathsf{srt}(1) \quad \phi_{\lambda,\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \quad \phi_{\mathsf{srt}(2),\mathsf{srt}(2)}(\mathsf{opn}(1)) = \mathsf{opn}(1)$$
$$\phi(\mathsf{srt}(3)) = \mathsf{srt}(2) \quad \phi_{\lambda,\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1) \quad \phi_{\mathsf{srt}(1)\cdot\mathsf{srt}(3),\mathsf{srt}(3)}(\mathsf{opn}(1)) = \mathsf{opn}(1)$$

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 $\{\mathsf{srt}(i) \mid n_2 < i \leqslant n_2'\}$, 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.

† Notice that, due to the heavy overloading of function symbols in abstract signatures, in order to describe morphisms, we need a decoration to distinguish the several components dealing with function symbols with respect to the arity.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Therefore, we get as extension

sig $\Sigma_1' =$
sorts $\mathsf{srt}(1), \mathsf{srt}(2), \mathsf{srt}(3), \mathsf{srt}(4)$
opns $\mathsf{opn}(1) : \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1) : \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \rightarrow \mathsf{srt}(1)$
$\mathsf{opn}(1) : \mathsf{srt}(1) \times \mathsf{srt}(3) \rightarrow \mathsf{srt}(3)$
$\mathsf{opn}(1) : \mathsf{srt}(2) \rightarrow \mathsf{srt}(2)$
$\mathsf{opn}(1),\ \mathsf{opn}(2) : \rightarrow \mathsf{srt}(4)$
$\mathsf{opn}(1) : \mathsf{srt}(3) \rightarrow \mathsf{srt}(4)$

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

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

using σ also to denote its pointwise extension to strings in **FOESign**. Also, let $em : \Sigma_2 \hookrightarrow \Sigma_2'$, where $\Sigma_2' = (S_2', OP_2', PR_2')$, be an admissible morphism in **YMon**.
Here and in the following, we will use $|X|$ to denote the cardinality of a set X.

— $\mathtt{ysig}(\phi, em) = (S, OP, PR)$, where:

– $S = \{\mathsf{srt}(i) \mid 1 \leqslant i \leqslant n\}$ 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 use $\overline{\sigma}$ to denote 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 $w \in S^*$, $s \in S$, $OP_{w,s} = \{\mathsf{opn}(i) \mid 1 \leqslant i \leqslant m\}$ 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 use $\overline{\psi}$ to denote 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 $w \in S^+$, $PR_w = \{\mathsf{prd}(i) \mid 1 \leqslant i \leqslant k\}$ 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 use $\overline{\pi}$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

to denote 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 taken 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.

(1) We have to show that $\mathtt{ymor}(\phi, em) \cdot \mathtt{ymon}(\phi, em) = em \cdot \phi$.
Let $s \in S_1$. Then $(\mathtt{ymor}(\phi, em) \cdot \mathtt{ymon}(\phi, em))(s) = \mathtt{ymor}(\phi, em)(\mathtt{ymon}(\phi, em)(s)) = \mathtt{ymor}(\phi, em)(s) = \overline{\sigma}(s) = \sigma(s) = (em \cdot \phi)(s)$.
The proof for operations and predicates is analogous.

(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), and $\mathtt{ysig}(\mathtt{ymor}(\phi, em), em')$ 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 $S = \{\mathsf{srt}(i) \mid 1 \leqslant i \leqslant n\}$ for $n = n_1 + (n_2'' - n_2)$, $n_1 = |S_1|$, $n_2'' = |S_2''|$ and $n_2 = |S_2|$.
Analogously, we have $\overline{S} = \{\mathsf{srt}(i) \mid 1 \leqslant i \leqslant \overline{n}\}$ 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 $\mathtt{ysig}(\phi, em)$, $S_1' = \{\mathsf{srt}(i) \mid 1 \leqslant i \leqslant n_1'\}$ 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 show that both yield 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 $s = \mathsf{srt}(i) \in S$

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

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

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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) This is trivial, as there is at most one admissible morphism between two signatures.

(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, Conditions 3b and 3c follow immediately by definition of $\mathtt{ymor}(\phi, Id_{\Sigma_2})$ and $\mathtt{ymon}(\phi, Id_{\Sigma_2})$.
But, by definition, $|S| = |S_1| + |S_2| - |S_2| = |S_1|$, and hence $S = S_1$. The required identification for operations and predicates can be proved analogously. □

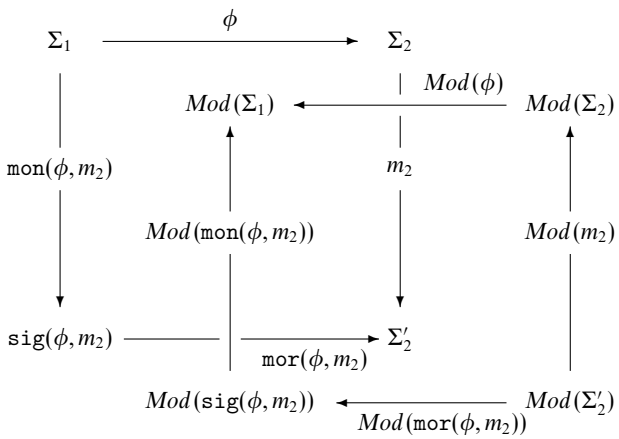
Let us now show that the properties required for local backward extensions are just that each $HMod(\phi)$ is a functor.

Proposition 2.10. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and $\mathtt{sig}$, $\mathtt{mon}$, $\mathtt{mor}$ be a local backward extension on a class **HMon** of admissible morphisms for $\mathscr{I}$.
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 $\langle A_2, m_2 \rangle \in |HMod(\Sigma_2)|$, that is, 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 recalls the syntactic counterpart):

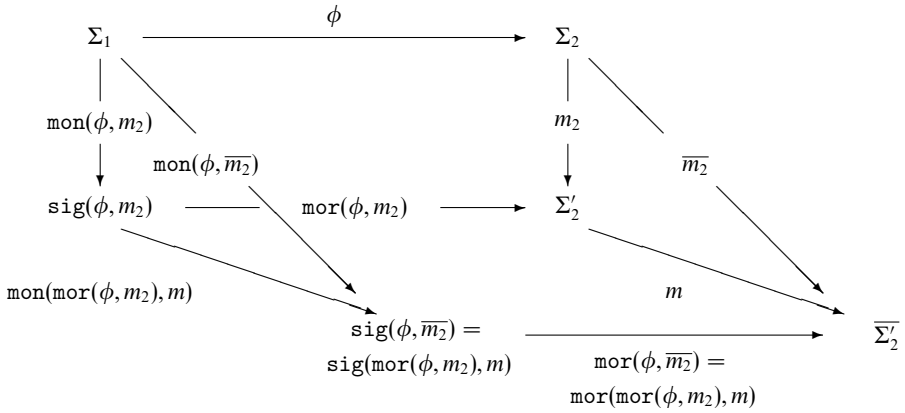


on morphisms:

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

for each $\langle p, m \rangle \in HMod(\Sigma_2)(\langle A_2, m_2: \Sigma_2 \hookrightarrow \Sigma'_2 \rangle, \langle \overline{A_2}, \overline{m_2}: \Sigma_2 \hookrightarrow \overline{\Sigma'_2} \rangle)$, which accords with the translation of models; a complexive picture is given below;

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press



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 $\langle p : A \to \overline{A_2}_{|m}, m : \Sigma_2' \hookrightarrow \overline{\Sigma_2'}\rangle$ 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 $\langle A_2, m : \Sigma_2 \hookrightarrow \Sigma_2'\rangle \in |HMod(\Sigma_2)|$

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

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 $HMod(\Sigma_2)$, and $\langle p, m\rangle$ 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 $HMod(\Sigma_2)$, we have $\langle q, m'\rangle \cdot \langle p, m\rangle = \langle q_{|m} \cdot p, m' \cdot m\rangle$ 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

$$HMod(\phi)(\langle q, m'\rangle) \cdot HMod(\phi)(\langle p, m\rangle)$$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 whose 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.

$$Mod(\mathtt{mor}(\phi, m_2))(q_{|m} \cdot p) = Mod(\mathtt{mor}(\phi, m_2))(q_{|m}) \cdot Mod(\mathtt{mor}(\phi, m_2))(p)$$

as $Mod(\mathtt{mor}(\phi, m_2))$ is a functor.
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

$$\begin{aligned} &Mod(\mathtt{mor}(\phi, \overline{m_2}) \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)) \quad = \\ &\qquad \text{(by Condition 2b, as } \overline{m_2} = m \cdot m_2) \\ &Mod(\mathtt{mor}(\mathtt{mor}(\phi, m_2), m) \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m)) \quad = \\ &\qquad \text{(by Condition 1)} \\ &Mod(m \cdot \mathtt{mor}(\phi, m_2)) \end{aligned}$$

Let us now consider the second components.

$$\begin{aligned} &\mathtt{mon}(\mathtt{mor}(\phi, \overline{m_2}), m') \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m) = \\ &\qquad \text{(by Condition 2b, as } \overline{m_2} = m \cdot m_2) \\ &\mathtt{mon}(\mathtt{mor}(\mathtt{mor}(\phi, m_2), m), m') \cdot \mathtt{mon}(\mathtt{mor}(\phi, m_2), m) = \\ &\qquad \text{(by Condition 2c (for } \phi = \mathtt{mor}(\phi, m_2))) \\ &\mathtt{mon}(\mathtt{mor}(\phi, m_2), m' \cdot m) \end{aligned}$$

□

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 a 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 by making a (not 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

$$\overline{\sigma}(s_1 \dots s_n) = \overline{\sigma}(s_1) \dots \overline{\sigma}(s_n)$$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

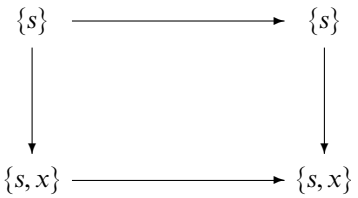
and

$$\overline{\sigma}(s) = \begin{cases} \sigma(s) & \text{if } s \in S_1 \\ s & \text{otherwise} \end{cases}$$

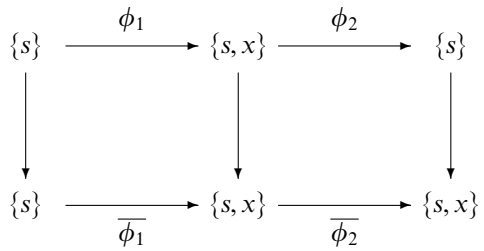
– $OP_{w,s} = (OP_1)_{w,s} \cup ((OP'_2)_{\overline{\sigma}(w),\overline{\sigma}(s)} - (OP_2)_{\overline{\sigma}(w),\overline{\sigma}(s)})$, for all $w \in S^*$, $s \in S$
– $PR_w = (PR_1)_w \cup ((PR'_2)_{\overline{\sigma}(w)} - (PR_2)_{\overline{\sigma}(w)})$, for all $w \in S^+$.

— $\mathtt{wmon}(\phi, em)$ 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 boring, is quite straightforward.
Then consider the following very simple case, where the signature has sorts only, and all arrows are plain embeddings,



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



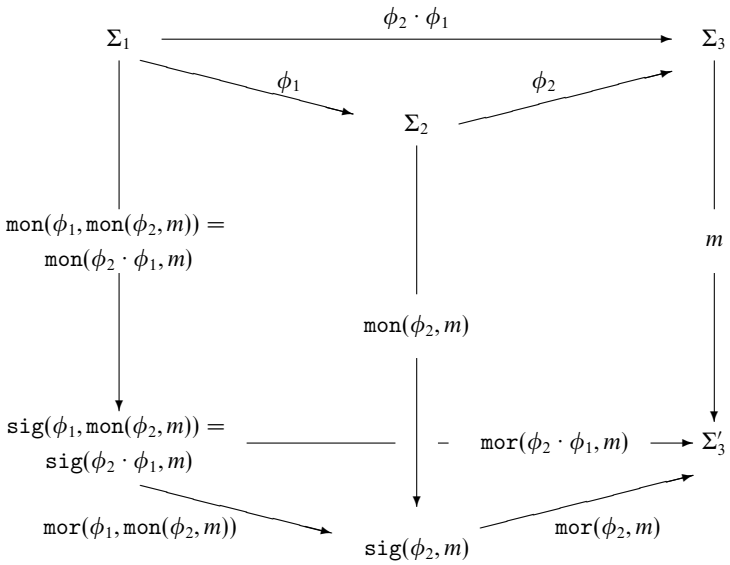
where $\overline{\phi_1}$ is the embedding and $\overline{\phi_2}(s) = \overline{\phi_2}(x) = s$.

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

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

4 The choice of $\mathtt{sig}$, $\mathtt{mor}$ and $\mathtt{mon}$ is natural with respect to 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) $\mathtt{mon}(\phi_2 \cdot \phi_1, m) = \mathtt{mon}(\phi_1, \mathtt{mon}(\phi_2, m))$.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press



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, suppose that all morphisms were acceptable, let us then 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 $\mathbf{Cat}^{\mathrm{Op}}$.

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

Proof. Because of Propositions 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, $HA = \langle A', m' : \Sigma \hookrightarrow \Sigma' \rangle$, $HB = \langle A'', m'' : \Sigma \hookrightarrow \Sigma'' \rangle$ belong to $HMod(\Sigma)$ and $\langle p, m \rangle : HA \to HB$ be a model morphism.

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

By analogy with the previous point, we can 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, let $HA = \langle A'_3, m'_3 : \Sigma_3 \hookrightarrow \Sigma'_3\rangle$ and $HB = \langle A'_3, m'_3 : \Sigma_3 \hookrightarrow \widetilde{\Sigma'_3}\rangle$ belong to $HMod(\Sigma_3)$ and let $\langle p, m : \Sigma'_3 \hookrightarrow \widetilde{\Sigma'_3}\rangle : HA \to HB$ be a model morphism.

on objects:

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

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:

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

Now let us consider the second component:

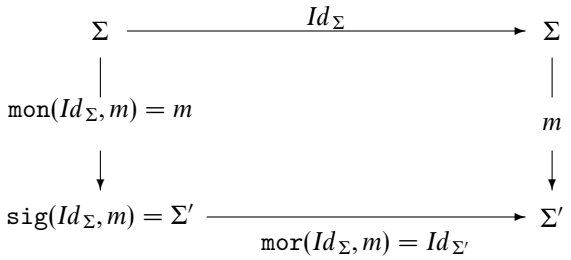
$$
\begin{array}{lll}
\mathtt{mon}(\mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)), \mathtt{mon}(\mathtt{mor}(\phi_2, m'_3), m)) & = & \text{by Condition 4c} \\
\mathtt{mon}(\mathtt{mor}(\phi_2, m'_3) \cdot \mathtt{mor}(\phi_1, \mathtt{mon}(\phi_2, m'_3)), m) & = & \text{by Condition 4b} \\
\mathtt{mon}(\mathtt{mor}(\phi_2 \cdot \phi_1, m'_3), m) & &
\end{array}
$$

□

It is worth noting that different backward extensions can be compatible with the same family of admissible morphisms. Let us look at an artificial but simple case.

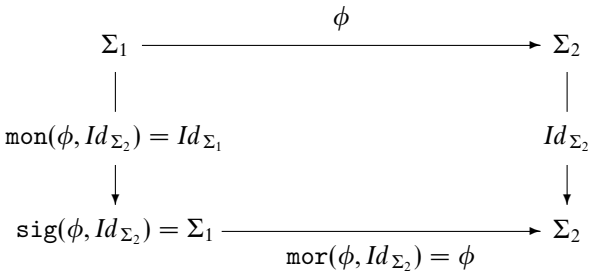
Example 2.14. Let **Sign** be a category with objects $\{X, Y, Z\}$ and as non-trivial arrows only $f : X \to Y$, $g : Y \to Z$ and their composition $h : X \to Z$.
As admissible monomorphisms we consider the identities g and h; since g and h are not composable, this class is closed under composition.
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 $\Sigma = Y$.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

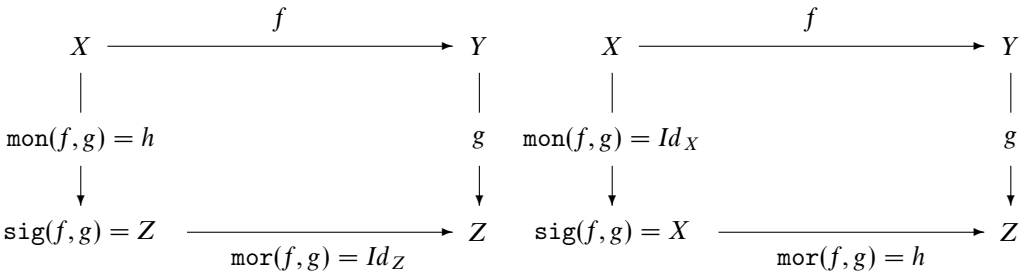


Moreover, Property 3 requires that backward extension along identities is the morphism itself; thus we have the following diagram, where we can have

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



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



It is straightforward (though 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 $\mathscr{I}$, but with very abstract models.

Proposition 2.15. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ 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 $\mathscr{I}$. Then

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

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

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

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $\models^H$, we have $\langle A, m \rangle \models^H_{\Sigma_2} Sen(\phi)(\xi)$ iff $A \models_{\Sigma_2'} Sen(m)(Sen(\phi)(\xi))$, that is, iff $A \models_{\Sigma_2'} Sen(m \cdot \phi)(\xi)$.

Since $\mathscr{I}$ is an institution, $A \models_{\Sigma_2'} Sen(m \cdot \phi)(\xi)$ iff $Mod(m \cdot \phi)(A) \models_{\Sigma_1} \xi$, that is, 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, which are actually 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 $\mathscr{H}(\mathscr{I})$ can be non-abstract algebraic (see *e.g.*, Tarlecki (1985) and Tarlecki(1986)) while $\mathscr{I}$ is so. Indeed, roughly speaking, the category $HMod(\Sigma)$ is the (disjoint) union of all categories $Mod(\Sigma')$ for some Σ' generalizing Σ, that is, such that 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), so $HMod(\Sigma_0)$ contains $Mod(\Sigma)$ for all signature Σ and hence is too large to be the terminal object in **Cat**, which 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 Burstall and 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, as 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 $\mathscr{FOE}$, with admissible morphisms and backward extensions as introduced in the following application.

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

Proof.

(4) Let $\phi_1: \Sigma_1 \to \Sigma_2$ and $\phi_2: \Sigma_2 \to \Sigma_3$ (with $\Sigma_1 = (S_1, OP_1, PR_1)$, $\Sigma_2 = (S_2, OP_2, PR_2)$, and $\Sigma_3 = (S_3, OP_3, PR_3)$), be morphisms in **FOESign**. Also, let $\iota: \Sigma_3 \hookrightarrow \Sigma_3'$, where $\Sigma_3' = (S_3', OP_3', PR_3')$, is an admissible morphism in **YMon**, which is a plain embedding. Moreover, let us use $\Sigma_2' = (S_2', OP_2', PR_2')$ to denote the signature $\mathtt{ysig}(\phi_2, \iota)$.

(4a) We have to show that the two 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.
We will show 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 $n_3' = |S_3'|$.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $\mathtt{ysig}(\phi_2, \iota)$, that is $n_2' = n_2 + n_3' - n_3$. Therefore, $|\overline{S}| = n_1 + n_2 + n_3' - n_3 - n_2 = |S|$.
We can show analogously 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 $\mathtt{ymor}(\phi_2 \cdot \phi_1, \iota) = \mathtt{ymor}(\phi_2, \iota) \cdot \mathtt{ymor}(\phi_1, \mathtt{ymon}(\phi_2, \iota))$.
We will show 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 part, we have $s = \mathsf{srt}(i)$ for some $1 \leqslant i \leqslant n_1 + n_3' - n_3$, and, by definition

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

and, analogously,

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

so that

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

as $n_1 < i \leqslant n_1 + n_3' - n_3$ implies $n_2 < i - n_1 + n_2$.

(4c) The proof is trivial, as there is at the most one injection between signatures.

(5) The proof is straightforward. □

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

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

called the institution of *hyper-loose specifications.*

Remark. Since the definition of admissible morphisms and the 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, which was presented in Application 2.16 for the case of many-sorted signatures (with predicates), applies, and hence, in most significant institutions and, in particular, it applies to the institutions of (conditional) equational specifications of many-sorted total (partial, non-strict) algebras.

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

Proof. The proof is trivial. □

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Thus, as ABSTRACT does not affect the signature category, admissible monomorphisms and backward extensions for an institution $\mathscr{I}$ are also admissible monomorphisms and backward extensions for any institution ABSTRACT($\mathscr{I}, \ldots$). Thus, several applications of ABSTRACT can be performed sequentially, given the ingredients only for the starting institution. In particular, we can always consider

$$\mathscr{I}' = \text{ABSTRACT}(\text{ABSTRACT}(\mathscr{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon}), \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon}),$$

which is, in general, different from ABSTRACT($\mathscr{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon}$) not only from a technical viewpoint (because the models of ABSTRACT($\mathscr{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon}$) are pairs $\langle A, m\rangle$, while the object of $\mathscr{I}'$ 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 **HMon** for an institution $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and a subcategory $\mathbf{Sign}'$ of **Sign**, with embedding E, the restriction of **HMon** to the elements of $\mathbf{Sign}'$ is a family of admissible monomorphisms for the institutions $\mathscr{I}' = (\mathbf{Sign}', Sen \cdot E, Mod \cdot E, \models)$. Moreover, the restriction of any backward extension `sig`, `mor` and `mon` for $\mathscr{I}$ such that the extensions of signature morphisms in $\mathbf{Sign}'$ along admissible monomorphisms in $\mathbf{Sign}'$ also yield morphisms in $\mathbf{Sign}'$, gives a backward extension for $\mathscr{I}'$.

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

Definition 2.18. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and let us use the following symbols $\xi, \xi_i \in Sen(\Sigma)$, $\Gamma, \Gamma' \subseteq Sen(\Sigma)$ and $A \in |Mod(\Sigma)|$.
An *entailment system* $\vdash$ for $\mathscr{I}$ is a family $\{\vdash_\Sigma \subseteq \wp(Sen(\Sigma)) \times Sen(\Sigma)\}_{\Sigma \in |\mathbf{Sign}|}$ 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 $\xi \in Sen(\Sigma)$;
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 $Sen(\phi)(\Gamma) \vdash_{\Sigma'} Sen(\phi)(\xi)$ for any $\phi\colon \Sigma \to \Sigma'$ in **Sign**.

An entailment is said to be *complete* iff the following condition holds:

completeness: if $A \models_\Sigma \xi$ for all A such that ($A \models_\Sigma \gamma$ 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 $A \in |Mod(\Sigma)|$.

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

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Proof. Since $\mathscr{I}'$ has the same signatures and sentences as $\mathscr{I}$ it is 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 $\mathscr{I}'$, 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 $\mathscr{I}$, 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 $\langle A, Id_\Sigma \rangle \models^H_\Sigma \xi$, which is $A \models_\Sigma \xi$. Therefore, $\Gamma \models^H_\Sigma \xi$ implies $\Gamma \models_\Sigma \xi$.
For the opposite direction, let us assume that $\Gamma \models_\Sigma \xi$ and let $\langle A, m : \Sigma \hookrightarrow \Sigma' \rangle \in |HMod(\Sigma)|$ 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 $Mod(m)(A) \models_\Sigma \xi$, which 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 to each other by a pair of functors. Now we will show that such a 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 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.

Proposition 2.20. Under the hypothesis and using the notation of Proposition 2.13, both $Emb : Mod \Rightarrow HMod$ and $Flat : HMod \Rightarrow Mod$, 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 $\phi : \Sigma \to \Sigma'$ be a signature morphism in **Sign** and consider the following diagram:

$$\begin{array}{ccc} Mod(\Sigma) & \xrightarrow{Emb_\Sigma} & HMod(\Sigma) \\ \uparrow & & \uparrow \\ Mod(\phi) & & HMod(\phi) \\ | & & | \\ Mod(\Sigma') & \xrightarrow[Emb_{\Sigma'}]{} & HMod(\Sigma') \end{array}$$

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*, we have

$$HMod(\phi)(\langle A, Id_{\Sigma'} \rangle)$$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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,

$$\langle Mod(\mathtt{mor}(\phi, Id_{\Sigma'}))(A), \mathtt{mon}(\phi, Id_{\Sigma'})\rangle = \langle Mod(\phi)(A), Id_{\Sigma}\rangle,$$

which is $Emb_{\Sigma} \cdot Mod(\phi)(A)$, by definition of *Emb*.

A similar proof applies to morphism translation. Indeed, let $p: A \to B$ be a morphism in $Mod(\Sigma')$. Then by definition of *Emb*, *HMod* and by Condition 3 of Definition 2.8:

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

and hence *Emb* is natural.

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

$$\begin{array}{ccc} Mod(\Sigma) & \xleftarrow{Flat_{\Sigma}} & HMod(\Sigma) \\ \uparrow & & \uparrow \\ Mod(\phi) & & HMod(\phi) \\ | & & | \\ Mod(\Sigma') & \xleftarrow[Flat_{\Sigma'}]{} & HMod(\Sigma') \end{array}$$

Let $\langle p, m\rangle$ be a homomorphism in $HMod(\Sigma')$ from $\langle A_1, m_1: \Sigma' \to \Sigma'_1\rangle$ into $\langle A_2, m_2: \Sigma' \to \Sigma'_2\rangle$.

Then $Mod(\phi) \cdot Flat_{\Sigma'}(\langle p, m\rangle)$ is

$$Mod(\phi)(Mod(m_1)(p)): Mod(\phi)(Mod(m_1)(A_1)) \to Mod(\phi)(Mod(m_2)(A_2)).$$

That is, $Mod(m_1 \cdot \phi)(p): Mod(m_1 \cdot \phi)(A_1) \to Mod(m_2 \cdot \phi)(A_2)$.

By definition of *HMod*, we have that $Flat_{\Sigma} \cdot HMod(\phi)(\langle p, m\rangle)$ 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,

$$Mod(\mathtt{mon}(\phi, m_1))(Mod(\mathtt{mor}(\phi, m_1))(p))$$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $Mod(m_1 \cdot \phi)(p): Mod(m_1 \cdot \phi)(A_1) \to Mod(m_2 \cdot \phi)(A_2)$ 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.

Definition 2.21. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and $\mathscr{I}' = (\mathbf{Sign}', Sen', Mod', \models')$ be institutions.

A *plain map of institution* (see, for example, Meseguer (1989)) $(\Phi, \alpha, \beta): \mathscr{I} \to \mathscr{I}'$ consists of

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

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, for example, Burstall and Goguen (1984) and Burstall and Goguen (1992)) $(\Phi, \alpha, \beta): \mathscr{I} \to \mathscr{I}'$ 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:

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

Proposition 2.22. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, **HMon** be a family of admissible morphisms, and `sig`, `mon` and `mor` be a backward extension on **HMon**. Moreover, let

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

be defined as in Proposition 2.15.

Then $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$ is a plain map of institutions from $\mathscr{H}$ into $\mathscr{I}$ and a morphism of institutions from $\mathscr{I}$ into $\mathscr{H}$ and $(Id_{\mathbf{Sign}}, Id_{Sen}, Flat)$ is a plain map of institutions from $\mathscr{I}$ into $\mathscr{H}$ and a morphism of institutions from $\mathscr{H}$ into $\mathscr{I}$.

Proof. To prove that $(Id_{\mathbf{Sign}}, Id_{Sen}, Emb)$ is a plain map of institutions from $\mathscr{H}$ into $\mathscr{I}$ and a morphism of institutions from $\mathscr{I}$ into $\mathscr{H}$, 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 $Mod(Id_\Sigma)(M) \models_\Sigma \xi$, since $Emb_\Sigma(M)$ is $\langle M, Id_\Sigma \rangle$.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Analogously, to prove that $(Id_{\mathbf{Sign}}, Id_{Sen}, Flat)$ is a plain map of institutions from $\mathscr{I}$ into $\mathscr{H}$ and a morphism of institutions from $\mathscr{H}$ into $\mathscr{I}$, 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 $\models^H$, we have $\langle M, m\rangle \models^H_\Sigma \xi$ iff $Mod(m)(M) \models_\Sigma \xi$, that is, by definition of *Flat*, iff $Flat_\Sigma(\langle M, m\rangle) \models_\Sigma \xi$. □

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 (Φ, α, β): $\mathscr{I} \to \mathscr{I}'$ be a map of institutions, $\mathscr{H}$ be $\mathsf{ABSTRACT}(\mathscr{I}, \ldots)$ and $\mathscr{H}'$ be $\mathsf{ABSTRACT}(\mathscr{I}', \ldots)$, 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 $\langle \widetilde{m}, \widetilde{m}\rangle$ such that $\widetilde{m}_{|\widetilde{m}} = A$.

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, its translation along *Emb* is initial for the very abstract models.

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

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

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 $\mathscr{I}$ and $\mathscr{H}$; let us denote by Mod the model class of T in $\mathscr{I}$, that is,

$$\mathsf{Mod} = \{A \mid A \in |Mod(\Sigma)| \textit{ such that } A \models_\Sigma \gamma \text{ for all } \gamma \in \Gamma\}$$

and by HMod the model class of T in $\mathscr{H}$, that is,

$$\mathsf{HMod} = \{HA \mid HA \in |HMod(\Sigma)| \textit{ such that } HA \models^H_\Sigma \gamma \text{ for all } \gamma \in \Gamma\}.$$

Since *Emb* and *Flat* preserve and reflect validity, by Proposition 2.22, they are also adjoint functors between Mod and HMod, by Proposition 2.7.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $HA = \langle A, m_A : \Sigma \hookrightarrow \Sigma' \rangle$ 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 $h : HA \to Emb(Flat(HA))$. 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 $\langle p, m : \Sigma' \hookrightarrow \Sigma \rangle$, we have $Id_{HA} = \epsilon_{HA} \cdot h = \langle Id_{(A_{|m_A})_{|m}} \cdot p, m_A \cdot m \rangle$, 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 $Emb(B) \in$ HMod and hence there is a unique

$$h : Emb(Flat(HA)) \to Emb(B),$$

so its image along *Flat* is a morphism and, since $Flat \cdot Emb$ is the identity,

$$Flat(h) : Flat(HA) \to B.$$

Moreover, if $k : Flat(HA) \to B$ is a morphism, then

$$Emb(k) : Emb(Flat(HA)) \to Emb(B),$$

and hence $Emb(k) = h$, so $k = Flat(Emb(k)) = Flat(h)$. 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, we have $A = Flat(Emb(A))$ is initial in Mod, that is, *Emb* reflects initiality. □

3. Enriching sentences by derived ones

3.1. *The operation* EXTEND

While the previous section dealt with a generalization of models (that is, with the semantic component of a formalism), here an operation is presented that allows the use of more expressive logic, but leaves 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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 is needed, in order to define the validity of the new sentences by a standard interpretation of the extra symbols. 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.

Proposition 3.1. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{Sign}_E$ be a subcategory of **Sign** with embedding $E : \mathbf{Sign}_E \to \mathbf{Sign}$, EXT be a functor from $\mathbf{Sign}_E$ into **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

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

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 $\mathscr{I}$ is an institution, we have, by the satisfaction condition for $\mathscr{I}$,

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

That is, since Ext is a natural transformation, iff $Ext_{\Sigma_1}(Mod(E(\phi))(A)) \models_{EXT(\Sigma_1)} \xi$, that is, 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, though 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, in order to prove that the result is actually an institution, it suffices to check that the definition of full abstraction of a model is compatible with the reducts along signature morphisms.

Let us look at 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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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. So, 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

$S \subseteq \vec{S}$, and

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

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

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

$\texttt{the}op \in \vec{OP}_{\lambda, \vec{s}}$ for each $op \in OP_{s_1 \ldots s_n, s}$ and $n > 0$, where

$$\vec{s} = (s_1 \times \ldots \times s_n \rightarrow s) \in \vec{S}\,.$$

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

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

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

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

† 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.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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

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

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

Extensions of models. Let A be a first-order structure on 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 $op \in OP$;

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

— $\texttt{apply}^{HA}(f, a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$ for each $\texttt{apply} \in \vec{OP}_{\vec{s}\, s_1 \ldots s_n, s}$, 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, $\mathscr{HO} = \mathsf{EXTEND}(\mathscr{FOE}, \mathbf{FOESign}_{HO}, EXT_{HO}, Ext_{HO})$ is an institution. An example of specification in $\mathscr{HO}$ taking advantage of the higher-order features is the specification of standard models of natural numbers:

spec *Standard* =
sorts *bool*, *nat*
opns *True*, *False* : → *bool*
Zero : → *nat*
S : *nat* → *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* → *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 (which 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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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.

Proposition 3.3. Under the hypothesis and using the notation of Proposition 3.1, let η be a natural transformation from E into EXT such that 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 $\mathsf{EXTEND}(\mathscr{I}, \mathbf{Sign}_E, EXT, Ext)$ into $\mathscr{I}$.

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 $\mathscr{I}$ is an institution, by the satisfaction condition for $\mathscr{I}$, $Ext_\Sigma(A) \models_{EXT(\Sigma)} Sen(\eta)(\xi)$ iff $Mod(\eta)(Ext_\Sigma(A)) \models_{E(\Sigma)} \xi$, that is, since $Ext_\Sigma(M)_{|\eta_\Sigma} = M$ for all models M, iff $A \models_{E(\Sigma)} \xi$. □

A simple example of an application of this construction is the introduction of equality in (fragments of either partial or total) first-order logic. In that case, if we are working with concrete signatures, 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, if we are 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 rationale behind describing institutions by means of operations on them is to make 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,

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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.

Proposition 3.4. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, $\mathbf{Sign}_E$ and $\mathbf{Sign}'_E$ be subcategories of **Sign** with embedding E and E', respectively. Moreover, let EXT and EXT' be functors into **Sign** from $\mathbf{Sign}_E$ and $\mathbf{Sign}'_E$, respectively. 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 in $\mathbf{Sign}'_E$, then $\mathscr{I}$, $\mathbf{Sign}_E$, $EXT' \cdot EXT$ and ${Ext'}_{EXT} \circ Ext$ are correct arguments for the operation EXTEND.

Proof. Since the image of EXT is contained in $\mathbf{Sign}'_E$, we have that $EXT' \cdot EXT$ 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 $\phi : \Sigma \to \Sigma'$ be a signature morphism in $\mathbf{Sign}_E$. Then $EXT(\phi) : EXT(\Sigma) \to EXT(\Sigma')$ is a signature morphism in $\mathbf{Sign}'_E$, because the image of EXT is included in $\mathbf{Sign}'_E$.
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:

$$\begin{array}{ccccc}
|Mod \cdot E(\Sigma)| & \xrightarrow{Ext_\Sigma} & \begin{array}{l}|Mod \cdot EXT(\Sigma)| = \\ |Mod \cdot E' \cdot EXT(\Sigma)|\end{array} & \xrightarrow{{Ext'}_{EXT(\Sigma)}} & |Mod \cdot EXT' \cdot EXT(\Sigma)| \\
\uparrow & & \uparrow & & \uparrow \\
|Mod \cdot E(\phi)| & & \begin{array}{l}|Mod \cdot EXT(\Sigma')| = \\ |Mod \cdot E' \cdot EXT(\phi)|\end{array} & & |Mod \cdot EXT' \cdot EXT(\phi)| \\
| & & | & & | \\
|Mod \cdot E(\Sigma')| & \xrightarrow[Ext_{\Sigma'}]{} & \begin{array}{l}|Mod \cdot EXT(\Sigma')| = \\ |Mod \cdot E' \cdot EXT(\Sigma')|\end{array} & \xrightarrow[{Ext'}_{EXT(\Sigma')}]{} & |Mod \cdot EXT' \cdot EXT(\Sigma')|
\end{array}$$

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

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 $\mathbf{Sign}_E$ such that its image along EXT is contained in $\mathbf{Sign}'_E$, without losing too many interesting languages.

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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, and 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 can apply EXTEND to some tuple $(\mathscr{I}, \mathbf{Sign}_E, EXT, Ext)$ and ABSTRACT to some tuple $(\mathscr{I}, \mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$, for the same institution $\mathscr{I}$. The compatibility conditions between $(\mathbf{Sign}_E, EXT, Ext)$ and $(\mathbf{HMon}, \mathtt{sig}, \mathtt{mor}, \mathtt{mon})$ that are 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 restricted in this way to $\mathbf{Sign}_E$, we only need that `sig`, `mor` and `mon` can be restricted as well, that is, that on extensible morphisms they also yield extensible morphisms. On the other hand, if we want first to generalize models, we need to be able to lift Ext to work on abstract models. To this end, 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).

Proposition 4.1. Let $\mathscr{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution, **HMon** be a family of admissible morphisms, and `sig`, `mon` and `mor` be a backward extension on **HMon**. Moreover, let $\mathbf{Sign}_E$ be a subcategory of **Sign** with embedding E, let EXT be a functor from $\mathbf{Sign}_E$ into **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 **HMon** is contained in the morphisms of $\mathbf{Sign}_E$, and EXT both preserves admissible monomorphisms and commutes with `sig`, `mon` and `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 hypotheses of both points are satisfied, the applications of EXTEND and

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

ABSTRACT in the two orders yield the same result, that is

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

Proof.

1 Since $\mathbf{Sign}_E$ is a subcategory of **Sign**, it is closed under morphism compositions and identities. Therefore, as **HMon** is closed under morphism compositions and identities too, $\mathbf{HMon}_E$ is also, 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 sig, mon and 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 show 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 with respect to mon, mor
$\langle Mod(\mathtt{mor}(EXT(\phi), EXT(m)))(Ext_{\Sigma_2'}(A)),$ $\mathtt{mon}(EXT(\phi), EXT(m))\rangle =$	by definition of $HMod(EXT(\phi))$
$HMod(EXT(\phi))(\langle Ext_{\Sigma_2'}(A), EXT(m)\rangle) =$ $HMod(EXT(\phi))(HExt_{\Sigma_2}(\langle A, m\rangle))$	by definition of $HExt$

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

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

where $HMod_E(\Sigma)$ consists of all pairs $\langle A, m: \Sigma \hookrightarrow \Sigma' \rangle$ with A an object of $Mod \cdot E(\Sigma') = Mod(\Sigma')$ and m an admissible morphism in $\mathbf{HMon}_E$. But, as **HMon** is contained in 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

$$\langle A, m: \Sigma \hookrightarrow \Sigma' \rangle \models_{\Sigma'} \xi \text{ iff } Mod(m)(A) \models^E_{\Sigma'} \xi \text{ iff } Ext_{\Sigma'}(Mod(m)(A)) \models_{EXT(\Sigma')} \xi.$$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Analogously,

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

Moreover, $\models''$ is defined by

$$\begin{aligned}\langle A, m\colon \Sigma \hookrightarrow \Sigma' \rangle \models''_\Sigma \ \xi \quad &\text{iff} \quad HExt_\Sigma(\langle A, m\rangle) \models^H_{EXT(\Sigma)} \xi \\ &\text{iff} \quad \langle Ext_\Sigma(A), EXT(m)\rangle \models^H_{EXT(\Sigma)} \xi \\ &\text{iff} \quad Mod(EXT(m))(Ext_\Sigma(A)) \models_{EXT(\Sigma')} \xi.\end{aligned}$$

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, $\mathscr{I}' = \mathscr{I}''$. □

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 a 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, which 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 $\mathscr{Y}$ 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, which we think is 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, such as commutativity). The practice of using very abstract specifications of abstract data types (VAS for short) 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 $\mathscr{Y}$, 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 (for example, sorts, operations, predicates, variables, terms, formulae, ...) and their interpretations. Let us now list the parameters for EXTEND.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 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 or 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 $\langle A_{|\phi}, Id_{\Sigma_2} \rangle$ (which 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, working in the opposite direction, 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 make terms of such extra sorts as close as possible to their 'meta' counterparts, we use 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 $n \in N$
$\lambda: \to var_seq$
$_ \cdot _: var \times var_seq \to var_seq$
$_: var \to term$
$\lambda: \to term_seq$
$_, _: term \times term_seq \to term_seq$
$_(_): pred \times term_seq \to atom$
$_ = _: term \times term \to atom$
$_: atom \to formula$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

$\neg\ _ : formula \to formula$
$_ \supset _ \wedge _ \vee _ \equiv _ : formula \times formula \to formula$
$\forall\ _ : _ . _ \exists\ _ : _ . _ : var_seq \times sort \times formula \to formula$

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 use $T_{\widetilde{\Sigma}}(\widetilde{X})$ to denote 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 $\widetilde{X}$:

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

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 $f \in OP_{w,s}$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

$Type^B(f) = s$ for all $f \in OP_{w,s}$
$PArity^B(p) = w$ for all $p \in PR_w$
$f^B = f^{T_\Sigma(\widetilde{X})}$ 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.

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

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

Let us now look at some examples of specifications in the institution $\mathscr{YFOE}$, 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 a module is allowed to devise a nice choice of extra constructors for trees, but 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
$\forall\, op : opn \,.\, Type(op) = tree \supset \exists\, t : term \,.\, HasSort(t, tree) \wedge Basic\text{-}gen(t) \wedge$
$\exists\, Var_1 \ldots Var_n : var \,.\, \exists\, s_1 \ldots s_n : sort \,.$
$Holds(\forall\, Var_1 : s_1 \,.\, \ldots \forall\, Var_n : s_1 \,.\, op(Var_1, \ldots, Var_n) = t)$
– each constructor for *sons* is derived
$\forall\, op : opn \,.\, Type(op) = sons \supset \exists\, t : term \,.\, HasSort(t, sons) \wedge Basic\text{-}gen(t) \wedge$
$\exists\, Var_1 \ldots Var_n : var \,.\, \exists\, s_1 \ldots s_n : sort \,.$
$Holds(\forall\, Var_1 : s_1 \,.\, \ldots \forall\, Var_n : s_1 \,.\, op(Var_1, \ldots, Var_n) = t).$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

For example the operation $1_{Ary}: state \times lab \times tree \rightarrow tree$ 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:

$$
\begin{aligned}
&Replac: tree \times lab \times tree \rightarrow tree \\
&Replac(T(s,sn),l,t) = T(s, Replac'(sn,l,t)) \\
&Replac': sn \times lab \times tree \rightarrow sn \\
&Replac'(\Lambda,l,t) = \Lambda \\
&Replac'(\langle l,t \rangle \ \& \ sn, l, t') = \langle l,t' \rangle \ \& \ Replac'(sn,l,t') \\
&l \neq l' \ \supset \ Replac'(\langle l,t \rangle \ \& \ sn, l', t') = \langle l,t \rangle \ \& \ Replac'(sn,l',t')
\end{aligned}
$$

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, which are (partial) functions yielding a specification, which 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 also, and an actual parameter is acceptable if it is a subclass of the formal parameter. Thus, the heading of a parameterized specification has the form $SP(X:VAS) = \ldots$ 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 constructors only is as follows:

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

Now, intuitively, we could instantiate $LIST$ on a specification of integer numbers in order to get lists of integers, but the names introduced by $LIST$, such as $list$ and Λ, should be unused in the actual parameter in order to avoid name clashes.

While many specification languages provide the 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
$\forall\, s: sort \,.\, s \neq list$

for $VAS0$ and be assured that no name clashes can happen during correct instantiations.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 $SET_FROM_LIST(X : VAS2) =$
enrich X **by**
axioms
$x::x::l = x::l$
$x_1::x_2::l = x_2::x_1::l$

where the parameter specification $VAS2$, in order 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 $\Lambda : \rightarrow list$
$_::_ : elem \times list \rightarrow list$
preds $equiv : list$
axioms
$\forall\, l, l' : list \,.\, equiv(l, l') \equiv equiv(l', l)$
$\forall\, l : list \,.\, equiv(l, l)$
$\forall\, l : list \,.\, \forall\, e : elem \,.\, \neg\, equiv(e::l, \Lambda)$
$\forall\, l : list \,.\, \forall\, e : elem \,.\, equiv(e::e::l, e::l)$
$\forall\, l : list \,.\, \forall\, x_1, x_2 : elem \,.\, equiv(x_1::x_2::l, x_2::x_1::l)$
$\forall\, w, w' : sort\, seq \,.\, \forall\, f : opn \,.\, Arity(f) = w \cdot list \cdot w' \supset$
$\forall\, \bar{t}, \bar{t'} : term_seq \,.\, HaveSorts(\bar{t}, w) \wedge HaveSorts(\bar{t'}, w') \supset$
$Holds(\forall\, Var_1, Var_2 : list \,.\, equiv(Var_1, Var_2) \supset (f(\bar{t}, Var_1, \bar{t'}) = f(\bar{t}, Var_2, \bar{t'})))$
$\forall\, w, w' : sort_seq \,.\, \forall\, p : pred \,.\, PArity(p) = w \cdot list \cdot w' \supset$
$\forall\, \bar{t}, \bar{t'} : term_seq \,.\, HaveSorts(\bar{t}, w) \wedge HaveSorts(\bar{t'}, w') \supset$
$Holds(\forall\, Var_1, Var_2 : list \,.\, equiv(Var_1, Var_2) \supset (p(\bar{t}, Var_1, \bar{t'}) \equiv p(\bar{t}, Var_2, \bar{t'})))$

The axioms of $VAS2$ prevent us instantiating SET_FROM_LIST with actual parameters, which could cause inconsistencies such as a specification of lists with predicates checking if a list has length n (a natural number).

Let us finally look at 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 the formal modelization of actor systems, C. Talcott has presented a class of structures, called *Abstract Actor Structures* (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 that can be completely different from those used in another system, and such values can also be built using the

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

'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 according with ρ. For example, if the values are lists of integers and of actor names, the application of such a renaming on a bijective mapping ρ and the 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 $f : s_1 \times \ldots \times s_n \to s$ 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 $Rename_{name}(\rho, n) = \rho(n)$.

We can use a very abstract specification to express all the above requirements, except 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

$$\mathscr{YFOEHO} = \mathsf{EXTEND}(\mathscr{YFOE}, \mathbf{FOESign}_{HO}, EXT_{HO}, Ext_{HO}),$$

where, with a slight notational abuse, we denote by $\mathbf{FOESign}_{HO}$ its restriction to **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 $\mathscr{YFOEHO}$. Let us look at the more significant part of such a 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** $message$
$_(_) : mapping \times name \to name$
$Rename : sort \to opn$
$Rename^* : mapping \times term_seq \to term_seq$
$\ldots$
axioms
$\forall\ s : sort\ .\ Arity(Rename(s)) = mapping, s$
$\forall\ s : sort\ .\ Type(Rename(s)) = s$
$Holds(\forall\ Var_1 : mapping\ .\ \forall\ Var_2 : name\ .\ Rename(name)(Var_1, Var_2) = Var_1(Var_2))$
$\forall\ \rho : mapping\ .\ Rename^*(\rho, \lambda) = \lambda$
$\forall\ s : sort\ .\ \forall\ \rho : mapping\ .\ \forall\ t : term\ .\ \forall\ wt : term_seq\ .$
$HasSort(t, s)\ \supset\ Rename^*(\rho, t, wt) = Rename(s)(\rho, t), Rename^*(\rho, wt)$
$\forall\ op : opn\ .\ \forall\ \bar{t} : term_seq\ .\ HaveSorts(\bar{t}, Arity(op))\ \supset$
$Holds(\forall\ Var_1 : mapping\ .\ Rename(Type(op))(Var_1, op(\bar{t})) = op(Rename^*(Var_1, \bar{t}))$
$\ldots$

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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 with it 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 a structure can be represented graphically 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 an algebra' (which is 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

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

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, where 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)†.

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

Under reasonable conditions, the conditional specifications in $\mathscr{E}$ admit initial models, 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.

† 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.

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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

First, we define

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

where **EMon** includes the elements of **YMon** that 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

$$\mathscr{V}\mathscr{E} = \mathsf{EXTEND}(\mathscr{Y}\mathscr{E}, \mathbf{VESign}, \mathit{COMPS}, \mathit{Comps})$$

where **VESign** is the subcategory of **ESign** such that it has the same objects and whose only morphisms are the isomorphisms. *COMPS* adds to an entity signature some predicates for testing what 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).

5. Conclusions and further work

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

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

It is interesting to note that the use of EXTEND is not limited to building very abstract institutions, but can be used also in other cases, see *e.g.*, Section 3 for the building of institutions for observational specifications and for second-order logic. Thus, operations on institutions could also provide nice tools for simplifying 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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

Cerioli and Reggio (1993), but these are case studies rather than an organic presentation of a 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 desired metalanguage for assembling formalisms. Some initial work (see Cerioli and Reggio (1995)) shows that a large number of institutions used in specification methods 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 (the last corresponds to using EXTEND).

Furthermore, after having determined an appropriate set of operations, we will have to study their properties, mainly with respect to 'moving among institutions', and their compositions so as 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, which 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. C. P. and Larsen, P. G. (eds.) Proc. FME'93: Industrial-Strength Formal Methods. *Springer-Verlag Lecture Notes in Computer Science* **670**.

Astesiano, E. and Reggio, G. (1993b) Specifying Reactive Systems by Abstract Events. *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. H., Poigné, A. and Rydeheard, D. E. (eds.) Proceedings of Category Theory and Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **283** 196–218.

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

Burstall, R. M. and Goguen, J. A. (1984) Introducing Institutions. In: Clarke, E. and Kozen, D. (eds.) Logics of Programming Workshop. *Springer-Verlag Lecture Notes in Computer Science* **164** 221–255.

Burstall, R. M. and Goguen, J. A. (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. (eds.) *Algebraic Methodology and Software Technology (AMAST'93)*, Workshops in Computing, Springer-Verlag.

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press

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.

Meseguer, J. (1992) Conditional Rewriting as a Unified Model of Concurrency. *Theoretical Computer Science* **96** 73–155.

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

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

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

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.

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. *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. *Theoretical Computer Science* **37** (3) 269–304.

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

https://doi.org/10.1017/S0960129597002466 Published online by Cambridge University Press