Institutions for Very Abstract Specifications*

M. Cerioli and G. Reggio

Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova
Viale Benedetto XV, 3 – 16132 Genova – Italy
e-mail: {cerioli, reggio}@disi.unige.it

Introduction

This paper is a first attempt at the definition of a set of operations on specification frameworks, supporting the modular construction of formal software specification methodologies.

Since obviously a formalism providing tools to deal with all possible software specification features, if any, would be a *monster* and would become out of date in a short time, in our opinion the first step, in order to produce formal specifications, is to get a framework including all the features needed by the particular problem under examination, but as simple as possible. Following an obvious reuse principle, the best way to get such a framework is to assemble pieces of formalisms, studied once and forever, or to tune an available formalism, adding only the "local" features.

As a simple example, think of lifting an existing algebraic approach, where just sentences without variables can be used to axiomatize the data types, to a richer formalism, where also formulae with variables are at hand.

In this paper, following the well-established approach by Goguen and Burstall (see e.g. [4, 5]), specification frameworks are formalized as *institutions*; thus enrichments and assembling of formalisms become, in this setting, operations among institutions.

The need for such a modular approach to the formalism construction has already been sporadically addressed in the literature. Consider for instance the *duplex institutions* by Goguen and Burstall [5], where an institution is built whose sentences comes from two input institutions, also applied in the database field by Fiadeiro and Reichwein [11]. Another example is the *extension by universal closure* by Sannella and Tarlecki [12], of a given institution and a set of its signature morphisms, where sentences are enriched along these signature morphisms regarding the extra-symbols as variables universally quantified. Moreover the *institution of implementation specifications* by Beierle and Voss [3] enrich an institution by tools to deal with implementation.

Here we present two more operations on institutions, that are originated by abstracting from the definition of several institutions for *very abstract specifications* in the field of concurrency: very abstract entity specifications [1], very abstract entity specifications with temporal logic [2], very abstract entity specifications with event

* This work has been partially supported by Esprit-BRA W.G. n.6112 Compass, Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R. (Italy), MURST-40% Modelli e Specifiche di Sistemi Concorrenti

logic [1, 9], each one in several variants, like with first-order, conditional, equational logic, with partial, non-strict, generalized models and so on.

Let us informally discuss the meaning of "very abstract". Every institution $\mathcal{I}$ defines some syntactic structures and offers a formal framework where to discuss, work on, and specify models on such syntaxes, giving an interpretation to the various syntactic elements (quite often algebras of some kind on signatures). Now we want to define a "very abstract institution", where to be able to express properties about not only the models on such syntaxes (i.e. about the interpretations), but also the syntaxes themselves (i.e. about the structure of the models). In this new setting we can characterize larger classes of models, having not all the same syntax, but whose syntaxes satisfy some conditions; thus we call specifications in this new institution very abstract.

An application of the very abstract operation in the field of abstract data types is the prove that the hyper-loose algebraic specifications by Pepper (see [7]), whose models on Σ are Σ'-algebras on some Σ' "extending" Σ, is an institution. A specification in this institution describe classes of algebras sharing a common syntax and satisfying properties on such common part, but with possibly some more structure, analogously to software realizations of a module, that are allowed to have, besides the operations required by the interface, further internal operations. However the sentences in this institution are the same as in the parameter institution, so that properties on the syntax of the specifications cannot be imposed. Extending also the sentences, here we get the institution of very abstract data types, which supports the specification of high-level requirements on modules also about their interfaces (e.g. constraints either in the number of operations or on the number of arguments of the operations, due to limits of the admissible implementations).

Let us consider, now, the institution of entity algebras, see [8], providing a formal framework for algebraic specifications of concurrent systems, where some signature operations are used to explicitly describe the concurrent structure (i.e. to define the system components, both static and dynamic, and the system architecture). Thus very abstract specifications (built on the entity institution) describe classes of entity algebras on possibly different signatures, i.e. formal models of systems with possibly different concurrent structures, satisfying common properties.

Besides these examples, the very abstract operation has many other interesting and useful concrete applications, already used in some industrial case studies of the specification, at different levels of abstraction, of a substation for the electric power distribution (see [10]).

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

The arguments of $\mathsf{ABSTRACT}$ are an institution and a family of signature "extensions", that are signature morphism satisfying some technical conditions. Thus, as the other parameters only depends on the signature category, the proof of the existence of such parameters can be shared by all institutions with the same syntactic part. In particular here we show the construction of signature extensions for the many-sorted signatures (with predicates), so that the same construction can be used for most "algebraic" institutions (e.g. institutions with partial or non-strict models and with every logic).

The paper is organized as follows. The operations ABSTRACT and EXTEND are introduced in Sect. 1 and 2, respectively. In Sect. 3 we apply them to get some interesting very abstract institutions, including the very abstract entity institution and the very abstract data types institution.

1 Abstracting Models w.r.t. Syntax

This section is devoted to the definition of an operation ABSTRACT that applied on a logical framework $\mathcal{I}$ yields a framework over $\mathcal{I}$, where more abstract specifications can be given, characterizing larger classes of models, that are required to provide a semantic counterpart for the specification syntax, but can have some extra-structure. A classical example of such a specification is the mathematical habit of regarding rings or fields as groups, without explicitly forgetting the product (the unity and the inverse) operations; indeed the models of the meta-specification "groups" need to have a group structure, but are as well allowed to have more operations. A more applicative example is the definition of software modules realizing a data type; indeed any such module is required to associate a function with each operation of the data type, but it is quite common in the practice to have (private) local definitions, different for every actual module, giving to the module an extra-structure.

1.1 The Parameters of ABSTRACT

Using the concept of *institution* (see e.g. [4, 5]) to represent logical frameworks, we define an operation on institutions yielding a new institution with the same syntax of its argument, i.e. with the same signatures and sentences, but whose models are, for each signature, the models of the original institution on each "extension" of the signature.

Let us recall the definition of institution (see e.g. [4]) and then introduce the basic ingredients needed to generalize the class of models on a signature.

Definition 1. An *institution* $\mathcal{I}$ consists of a category **Sign** of *signatures*, a functor $Sen\colon \mathbf{Sign} \to \mathbf{Set}$ giving the set of *sentences* over a signature, a functor $Mod\colon \mathbf{Sign} \to \mathbf{Cat}^{\mathbf{Op}}$ 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 $\phi\colon \Sigma \to \Sigma'$ 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 $Sen(\Sigma)$. □

Let us fix an institution $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and discuss the parameters for getting a institution $\mathcal{H} = \mathsf{ABSTRACT}(\mathcal{I}, \ldots)$ based on $\mathcal{I}$. These elements and their properties are summarized in tables enclosed by boxes.

Intuitively in $\mathcal{H}$ a signature Σ represents the minimal structure that its models have, but the models can have a richer structure than the one explicitly described by Σ; thus the Σ-models in $\mathcal{H}$ are the Σ'-models in $\mathcal{I}$, for some Σ' which "extends" 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 morphisms; this leads to consider the class of these morphisms, called *admissible*, as one of the ABSTRACT parameters. Note that two minimal requirements have to be imposed on this class: that the identities are admissible, corresponding to the intuition that each signature is the trivial extension of itself, and that the class of admissible morphisms is closed under composition, because extending an extension should result in an extension, too.

$$\mathbf{HAMor} = \{\mathbf{HAMor}(\Sigma, \Sigma')\}_{\Sigma,\Sigma' \in |\mathbf{Sign}|} \text{ s.t. for all } \Sigma, \Sigma', \Sigma'' \in |\mathbf{Sign}|$$

- $\mathbf{HAMor}(\Sigma, \Sigma')$ is a (possibly empty) set of morphisms from Σ into Σ';
- $Id_\Sigma \in \mathbf{HAMor}(\Sigma, \Sigma)$ (identities are admissible morphisms)
- if $m \in \mathbf{HAMor}(\Sigma, \Sigma')$ and $m' \in \mathbf{HAMor}(\Sigma', \Sigma'')$, then $m' \cdot m \in \mathbf{HAMor}(\Sigma, \Sigma'')$

Admissible morphisms

Here and in the following we write $m\colon \Sigma \hookrightarrow \Sigma'$ to denote that m is an admissible morphism from Σ into Σ'; 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 $\mathbf{HAMor}(\Sigma, \Sigma')$.

It is now possible to define the abstract models on any signature Σ, that are pairs $<A, m>$, for $m\colon \Sigma \hookrightarrow \Sigma'$ and $A \in |Mod(\Sigma')|$; note that we need to keep track of the way Σ' extends Σ, because in general Σ' may be an extension of Σ in different ways, as several morphisms with the same domain and codomain can be admissible.

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

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

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

There is (at least) another natural choice of the model morphisms in $HMod(\Sigma)$, that is to have as morphisms from $<A, m_1: \Sigma \hookrightarrow \Sigma'>$ into $<B, m_2: \Sigma \hookrightarrow \Sigma''>$ the Σ-morphisms in $Mod(\Sigma)$ from $A_{|m_1}$ into $B_{|m_2}$. However in this way the morphisms do not depend on the extra-structure of Σ' and Σ'', and hence two abstract models, $<A, m>$ and $<B, m>$ on the same extension can be isomorphic if their restriction along m are such, while A and B are not even related by a homomorphism either way. Instead, following our choice, two abstract models (on the same extension) are isomorphic in the new institution iff they are isomorphic in the starting institution, accordingly with the intuition that the nature of the specified models is the same and in the new institution we are only able to specify "bigger" classes of original models.

Definition 2. For every $\Sigma \in |\mathbf{Sign}|$, the category $HMod(\Sigma)$ is defined by:

Objects $|HMod(\Sigma)| = \{<A, m> \mid m \in \mathbf{HAMor}(\Sigma, \Sigma'), A \in |Mod(\Sigma')|\}$;
Morphisms $HMod(\Sigma)(<A, m_1: \Sigma \hookrightarrow \Sigma'>, <B, m_2: \Sigma \hookrightarrow \Sigma''>) =$
$\{<p, m> \mid m \in \mathbf{HAMor}(\Sigma', \Sigma''), m \cdot m_1 = m_2 \text{ and } p \in Mod(\Sigma')(A, B_{|m})\}$;
Identities $Id_{<A, m: \Sigma \hookrightarrow \Sigma'>} = <Id_A, Id_{\Sigma'}>$;
Composition $<p, m: \Sigma' \hookrightarrow \Sigma''> \cdot <q, m': \Sigma'' \hookrightarrow \Sigma'''> = <p \cdot q_{|m}, m' \cdot m>$. □

A graphical view of this composition is given below.

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

Proposition 3. *For every $\Sigma \in |\mathbf{Sign}|$, $HMod(\Sigma)$ is a category.* □

In order to generalize the above definition of $HMod(\Sigma)$ to a functor on **Sign**, a family of functors $HMod(\phi)$, for every $\phi \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$, has to be defined, preserving identities and composition. To define such any $HMod(\phi)$, since the Σ_2-objects in $\mathcal{H}$ are $\mathcal{I}$-models on extensions of Σ_2, we need a uniform way of building extensions of Σ_1 starting from the Σ_2-extensions. Thus in the following definition we introduce a new ingredient for the $\mathcal{H}$-construction.

It is worth to note that the conditions 1 and 2 are needed for $HMod(\phi)$ to be a functor, while 3, 4 and 5 are needed for $HMod$ to be a functor, too, i.e. that preserves composition and identities in **Sign**.

Definition 4. A *backward extension* on a class **HAMor** of admissible morphisms in **Sign**, 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 $\mathtt{amor}(\phi, m): \Sigma_1 \hookrightarrow \mathtt{sig}(\phi, m)$ for each signature morphism

$\phi\colon \Sigma_1 \to \Sigma_2$ and each admissible morphism $m\colon \Sigma_2 \hookrightarrow \Sigma_2' \in \mathbf{HAMor}$ satisfying the following conditions [2]:

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

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

2. The choice of `sig`, `mor` and `amor` is natural w.r.t. the second argument:
   (a) $\texttt{sig}(\phi, m' \cdot m) = \texttt{sig}(\texttt{mor}(\phi, m), m')$;
   (b) $\texttt{mor}(\phi, m' \cdot m) = \texttt{mor}(\texttt{mor}(\phi, m), m')$;
   (c) $\texttt{amor}(\phi, m' \cdot m) = \texttt{amor}(\texttt{mor}(\phi, m), m') \cdot \texttt{amor}(\phi, m)$;

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

3. The choice of `sig`, `mor` and `amor` is natural w.r.t. the first argument:
   (a) $\texttt{sig}(\phi_2 \cdot \phi_1, m) = \texttt{sig}(\phi_1, \texttt{amor}(\phi_2, m))$;
   (b) $\texttt{mor}(\phi_2 \cdot \phi_1, m) = \texttt{mor}(\phi_2, m) \cdot \texttt{mor}(\phi_1, \texttt{amor}(\phi_2, m))$;
   (c) $\texttt{amor}(\phi_2 \cdot \phi_1, m) = \texttt{amor}(\phi_1, \texttt{amor}(\phi_2, m))$;

[2] Note that the (a)-conditions at points 2, 3, 4 and 5 folllow from the corresponding (b)-conditions (and/or from the (c)-conditions) and are mentioned for the sake of clearness.

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

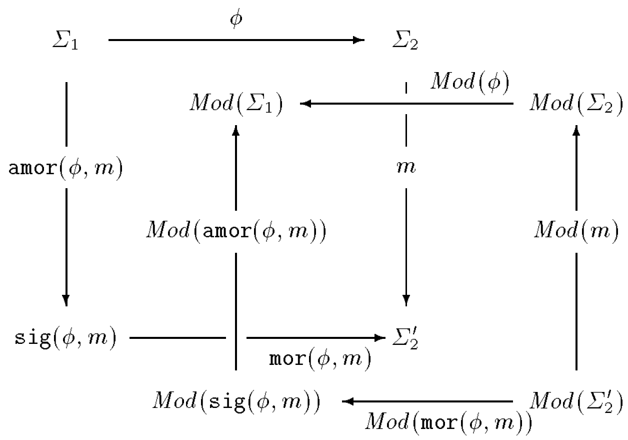
4. The identity as first argument is preserved:
   (a) $\mathtt{sig}(Id_{\delta_0(m)}, m) = \delta_1(m)$;
   (b) $\mathtt{mor}(Id_{\delta_0(m)}, m) = Id_{\delta_1(m)}$;
   (c) $\mathtt{amor}(Id_{\delta_0(m)}, m) = m$.
5. The identity as second argument is preserved:
   (a) $\mathtt{sig}(\phi, Id_{\delta_1(\phi)}) = \delta_0(\phi)$;
   (b) $\mathtt{mor}(\phi, Id_{\delta_1(\phi)}) = \phi$;
   (c) $\mathtt{amor}(\phi, Id_{\delta_1(\phi)}) = Id_{\delta_0(\phi)}$. □

Since any backward extension on **HAMor** is sufficient to define a model functor, a backward extension is the last parameter of the operation we are describing.

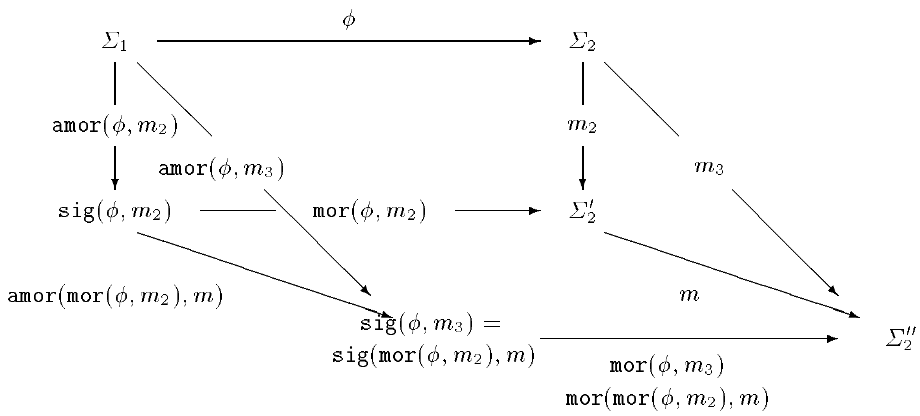
$\mathtt{sig}, \mathtt{amor}, \mathtt{mor}$ Backward extensions on **HAMor**

Proposition 5. *Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution and* $\mathtt{sig}$*,* $\mathtt{amor}$*,* $\mathtt{mor}$ *be a backward extension on a class* **HAMor** *of admissible morphisms. 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*
  $HMod(\phi)(<A, m: \Sigma_2 \hookrightarrow \Sigma_2'>) =$
  $<Mod(\mathtt{mor}(\phi, m))(A), \mathtt{amor}(\phi, m): \Sigma_1 \hookrightarrow \mathtt{sig}(\phi, m)>$
  for every $<A, m> \in |HMod(\Sigma_2)|$, i.e. the admissible morphism m is translated into the admissible morphism provided by the backward extensions and the model A is accordingly translated along the (model-interpretation of) the extension of ϕ, as the front side of the following picture shows (the back side reminds the syntactic counterpart):



– *on morphisms*
$HMod(\phi)(<p, m>) = <Mod(\mathtt{mor}(\phi, m_2))(p), \mathtt{amor}(\mathtt{mor}(\phi, m_2), m)>$
for every $<p, m> \in HMod(\Sigma_2)(<A, m_2 \colon \Sigma_2 \hookrightarrow \Sigma_2'>, <B, m_3 \colon \Sigma_2 \hookrightarrow \Sigma_2''>)$*, accordingly with the translation of models; a complexive picture is given below;*



Then $HMod(\phi)$ *is a functor.* □

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

Proposition 6. *Under the hypothesis and using the notation of proposition 5, HMod is a functor from* **Sign** *into* $\mathbf{Cat}^{\mathbf{Op}}$. □

As the sentences of the result institution are the same as the one of the parameter institution, the validity relation for "new" models can be defined in terms of the "old" validity relation and hence we do not need any other parameter in order to define the ABSTRACT operation.

1.2 The ABSTRACT Operation

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

where **HAMor** is a family of admissible morphisms, `sig`, `amor` and `mor` are a backward extension on **HAMor**, $HMod$ is given as in Prop. 6, $\models^H$ is defined by:
for each model $<A, m \colon \Sigma \hookrightarrow \Sigma'>$ in $|HMod(\Sigma)|$ and each ξ in $Sen(\Sigma)$

$$<A, m> \models^H_\Sigma \xi \iff A \models_{\Sigma'} Sen(m)(\xi) \iff A_{|m} \models_\Sigma \xi.$$

The ABSTRACT operation

Proposition 7. *Let* $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ *be an institution,* **HAMor** *be a family of admissible morphisms, and* `sig`, `amor` *and* `mor` *be a backward extension on* **HAMor**. *Then* $\mathsf{ABSTRACT}(\mathcal{I}, \mathbf{HAMor}, \mathtt{sig}, \mathtt{amor}, \mathtt{mor})$ *is an institution.* □

Remark. Note that, besides the institution $\mathcal{I}$ itself, the arguments of the ABSTRACT operation only depend on the signature category; thus the "non-canonical" part, i.e. the choice of **HAMor**, `sig`, `amor` and `mor`, can be shared by the institutions with the same signatures, disregarding the models and the sentences. In particular in section 3 a choice for `sig`, `amor` and `mor` backward extension on the embeddings as admissible morphisms is presented in the case of many-sorted signatures (with predicates), that applies, hence, in most significant institutions.

2 Enriching sentences by derived ones

Another useful operation to modularly build institutions consists of enriching the sentences by regarding as sentences on a signature Σ the sentences on a larger signature $EXT(\Sigma)$. A canonical example is adding equality to first-order logic, by coding the equality as a predicate.

As in general we may able to enrich only the sentences built over particular signatures and the "new" sentences may be incompatible with some "old" signature morphisms, a subcategory of signatures has to be selected as signature category of the result of this operation.

On the semantic side a canonical way of extending the models, in order to define the validity of the new sentences by a standard interpretation of the extra-symbols, is needed: this extension is given by a natural transformation Ext transforming models on Σ into models on $EXT(\Sigma)$, disregarding the model morphisms that are not involved in the definition of validity.

Let us summarize the arguments and the result of this operation.

Given an institution $\mathcal{I}$, a subcategory $\mathbf{Sign}_E$ of $\mathbf{Sign}$ with embedding $E\colon \mathbf{Sign}_E \to \mathbf{Sign}$, a functor $EXT\colon \mathbf{Sign}_E \to \mathbf{Sign}$ and a natural transformation $Ext\colon set \cdot Mod \cdot E \to set \cdot Mod \cdot EXT$ (set is the functor dropping from a category all morphisms different from the identities)

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

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

The EXTEND operation

Proposition 8. *Let $\mathcal{I}$, $\mathbf{Sign}_E$, EXT and Ext be as in the above table; then $\mathsf{EXTEND}(\mathcal{I}, \mathbf{Sign}_E, EXT, Ext)$ is an institution.* □

3 Examples

3.1 The institution of hyper-loose specifications

Let $\mathcal{FOE} = (\mathbf{FOESign}, FOESen, FOEMod, \models^{FOE})$ denote the institution of many-sorted first-order logic with equality; in the sequel we assume that **FOESign** is the category of *abstract* signatures, i.e. the quotient of the usual category of first-order signatures w.r.t. isomorphisms, so that Σ stands for the class of signatures isomorphic to Σ.

The institution of the hyper-loose (many-sorted first-order with equality) specifications, introduced in [7], is the very abstract institution over $\mathcal{FOE}$ defined below using only the operation $\mathsf{ABSTRACT}$; we do not use EXTEND in this case, since we do not need to extend the sentences.

Following Sect. 1 we give the parameters for the operation $\mathsf{ABSTRACT}$.
Morphisms characterizing the signature extensions. The elements of **YAMor** are the embeddings between many-sorted signatures, i.e.:

$$\mathbf{YAMor}_{\Sigma,\Sigma'} = \{em\colon \Sigma \hookrightarrow \Sigma' \mid em \text{ is injective}\}.$$

Backward extension. Let $\phi\colon \Sigma_1 \to \Sigma_2$ be a morphism in **FOESign** and $em\colon \Sigma_2 \hookrightarrow \Sigma_2'$ be an admissible morphism in **YAMor**. The idea 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[3].

$$\begin{tikzcd}
\boxed{\Sigma_1} \arrow[r, "\phi"] \arrow[d] & \boxed{\Sigma_2} \arrow[d, "em"] \\
\boxed{\Sigma_1} \arrow[r, "\phi"] & \boxed{\Sigma_2} \\
\boxed{\Sigma_2' - \Sigma_2} \arrow[r, "Id"'] & \boxed{\Sigma_2' - \Sigma_2}
\end{tikzcd}$$

[3] Here, as in the sequel, we assume that representative of different signature classes do not share any symbol, so that the union of Σ_1 and $\Sigma_2' - \Sigma_2$ is disjoint.

- $\texttt{ysig}(\phi, em) = (S, OP, PR)$, where:
  * $S = Sorts(\Sigma_1) \cup (Sorts(\Sigma'_2) - Sorts(\Sigma_2))$
  * For all $s_1, \ldots, s_n, s_{n+1} \in S$
    $$OP_{s_1 \ldots s_n, s_{n+1}} = Opns(\Sigma_1)_{s_1 \ldots s_n, s_{n+1}} \cup \bigcup_{s'_i = \overline{\phi}(s_i),\ i=1,\ldots,n+1} (Opns(\Sigma'_2)_{s'_1 \ldots s'_n, s'_{n+1}} - Opns(\Sigma_2)_{s'_1 \ldots s'_n, s'_{n+1}})$$
    where $\overline{\phi}(s) = \phi(s)$ if $s \in Sorts(\Sigma_1)$, s otherwise.
  * For all $s_1, \ldots, s_n \in S$
    $$PR_{s_1 \ldots s_n} = Preds(\Sigma_1)_{s_1 \ldots s_n} \cup \bigcup_{s'_i = \overline{\phi}(s_i),\ i=1,\ldots,n} (Preds(\Sigma'_2)_{s'_1 \ldots s'_n} - Preds(\Sigma_2)_{s'_1 \ldots s'_n}).$$

  It is obvious to see that $\texttt{ysig}(\phi, em)$ is a many-sorted signature.
- $\texttt{yamor}(\phi, em)$ is the embedding of Σ_1 into $\texttt{ysig}(\phi, em)$, i.e. $\texttt{yamor}(\phi, em)(s) = s$, $\texttt{yamor}(\phi, em)(Op) = Op$, $\texttt{yamor}(\phi, em)(Pr) = Pr$, for all s, Op, Pr.
- $\texttt{ymor}(\phi, em)$ is the morphism from $\texttt{ysig}(\phi, em)$ into Σ'_2 is defined as follows:
  * on sorts:
    $\texttt{ymor}(\phi, em)(s) = em(\phi(s))$, if $s \in Sorts(\Sigma_1)$, s otherwise;
  * on operations:
    $\texttt{ymor}(\phi, em)(Op) = em(\phi(Op))$ if Op is in $Opns(\Sigma_1)$, Op otherwise;
  * on predicates:
    $\texttt{ymor}(\phi, em)(Pr) = em(\phi(Pr))$ if Pr is in $Preds(\Sigma_1)$, Pr otherwise.

Proposition 9. ysig, yamor *and* ymor *defined above are a backward extension on* **YAMor**. □

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

The institution of hyper-loose specifications

3.2 An institution for the very abstract specifications of data types

The idea is to extend the institution $\mathcal{Y}$ with appropriate formulae for expressing requirements on the (extra part of the) signatures of the very abstract models. It is obvious that there are different ways to choose these requirements; here we present a rather general and powerful choice, that we think appropriate for many reasonable applications. Our idea is to give the possibility to express both purely syntactic conditions on the extra part of the models (e.g. requiring the (non) existence of an operation or a predicate whose functionality satisfies some conditions) but also semantic one (e.g. requiring the (non) existence of an operation or a predicate whose interpretation satisfies some conditions, as commutativity). The practice of using very abstract specifications of abstract data types (shortly VAS) will show whether this choice is appropriate possibly suggesting improvements and modifications.

Then the institution for VAS is built by applying the operation EXTEND of Sect. 2 to the institution $\mathcal{Y}$ and the underlying idea is to take as new formulae on a signature Σ the formulae of classical first-order logic with equality on the signature enriching Σ by sorts, operations and predicates for handling the syntactic elements on Σ (e.g.: sorts, operations, predicates, variables, terms, formulae, ...) and their interpretations.

Now we list the parameters for EXTEND.

The extendible signatures. The above kind of sentence extension may be done on each many-sorted signature, so we do not restrict the objects of **FOESign**; however we have to restrict the admissible signature morphisms. For example let θ be the formula "$\exists x, y\colon sort\,.\, x \neq y$", Σ_1 be the signature having just one sort, srt, Σ_2 be the signature having two sorts, respectively srt_1 and srt_2, no operations and no predicates and $\phi\colon \Sigma_1 \to \Sigma_2$ be the signature morphism defined by $\phi(srt_1) = srt = \phi(srt_2)$. Now θ is false on all algebras in $YMod(\Sigma_1)$, while its translation along ϕ holds on the algebras in $YMod(\Sigma_2)$.

VSign is the category whose objects are the same of **FOESign** and whose only morphisms are the isomorphisms of **FOESign** and $VE\colon$ **VSign** $\to$ **FOESign** is the embedding functor.

The extended signatures. $SYNT\colon$ **VSign** $\to$ **VSign** is the functor defined by:

- on objects:

  $$\begin{array}{l}
SYNT((S, OP, PR)) = \\
(S \cup \quad \{sort, atom, formula,\} \cup \{opn_{w,s}\}_{\substack{w\in S^*, s\in S\\ OP_{w,s}\neq\emptyset}} \cup \{pred_w\}_{\substack{w\in S^*\\ PR_w\neq\emptyset}} \\
\cup \quad \{seq_w\}_{\substack{w\in S^*\\ PR_w\neq\emptyset \text{or} OP_{w,s}\neq\emptyset}} \cup \{var_s\}_{s\in S} \cup \{term_s\}_{s\in S} \cup \ldots \\
\;OP \cup \{s\colon \to sort \mid s \in S\} \cup \{Op\colon \to opn_{w,s} \mid Op \in OP_{w,s}, w \in S^*, s \in S\} \\
\cup \quad \{Pr\colon \to pred_w \mid Pr \in PR_w, w \in S^*\} \cup \\
\cup \quad \{_(_)_{w,s}\colon opn_{w,s} \times seq_w \to term_s \mid w \in S^*, s \in S\}, \cup \ldots \\
\;PR \cup \{Holds\colon formula, \ldots\})
\end{array}$$

  For lack of room we do not give the complete definition of the extended signature, however it is easy to understand how to complete it with the remaining sorts, operations and predicates.
- on morphisms:
  the obvious extension leaving the new symbols inaffected
  $SYNT(\phi)(\sigma) = \phi(\sigma)$ if $\sigma \in \Sigma$, otherwise $SYNT(\phi)(\sigma) = \sigma$.

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

Interpretation of the extended signatures. For all $\Sigma \in |set(YMod(\mathbf{VSign}))|$ $Synt_\Sigma\colon |YMod(\Sigma)| \to YMod(SYNT(\Sigma))$ is the functor defined by:

- on objects:
  if $A \in |\mathbf{YAMor}(\Sigma)|$, i.e. A is a Σ_1-algebra where $\Sigma_1 = (S, OP, PR)$, then, fixed for every sort $s \in S$ a denumerable set of variables X_s, $Synt_\Sigma(A) = B$, where
  $B_{sort} = S$, $B_{vars_s} = X_s$, $B_{opn_{w,s}} = OP_{w,s}$, $B_{pred_w} = PR_w$,
  $B_{formula}$ = the set of all first-order formulae on Σ_1 and X,
  ...
  $op^B = op^A$ if $op \in OP_{w,s}$ for some w, s,
  ...
  $Holds^B(\theta) = A \models^{FOE} \theta$,
  ...

– on morphisms:
  Obvious.

Since the morphisms in **VSign** are isomorphisms, obviously *Synt* is a natural transformation.

Example 1. We specify the fundamental requirements on a module for handling labelled transition trees without completely fixing the interface. The designer in charge of realizing such module is allowed to devise a nice choice of extra constructors for trees, but it cannot add operations modifying parts of a tree, so that it is possible to give implementations where repeated common subtrees are shared.

spec LTT =
  enrich $LAB, STATE$ **by**
  sorts $tree, sons$
  opns
  – fixed components of the interface
    $\Lambda\colon \to sons$
    $<\_,\_>\&\_\colon lab \times tree \times sons \to sons$
    $T\colon state \times sons \to tree$
  axioms
  – properties of the fixed part of the interface
    $<l,t>\&<l,t>\&sons = <l,t>\&sons$
    $<l_1,t_1>\&<l_2,t_2>\&sons = <l_2,t_2>\&<l_1,t_1>\&sons$
  – properties of the variable part of the interface
    – each constructor for $tree$ is derived
    $\forall op\colon opn_{w,tree} \,.\, (\exists t : term_{tree} \,.\, Holds(op(x_1,\ldots,x_n) = t) \wedge$
      $Operations(t) \subseteq \{<\_,\_>\&\_, T\} \cup Opns(STATE) \cup Opns(LAB))$
    – each constructor for $sons$ is derived
    $\forall op\colon opn_{w,sons} \,.\, (\exists t : term_{sons} \,.\, Holds(op(x_1,\ldots,x_n) = t) \wedge$
      $Operations(t) \subseteq \{<\_,\_>\&\_, T\} \cup Opns(STATE) \cup Opns(LAB))$

For example the operation $1_{Ary}\colon state \times lab \times tree \to tree$ building unary trees and defined by $1_{Ary}(s,l,t) = T(s,<l,t>\&\Lambda)$ can be added to the interface, while the following one, replacing some subtrees, cannot:

$$
\begin{array}{l}
Replac\colon tree \times lab \times tree \to tree \\
Replac(T(s,sns),l,t) = T(s, Replac'(sns,l,t)) \\
Replac'\colon sns \times lab \times tree \to sns \\
Replac'(\Lambda,l,t) = \Lambda \\
Replac'(<l,t>\&sns,l,t') = <l,t'>\&Replac'(sns,l,t') \\
l \neq l' \supset Replac'(<l,t>\&sns,l',t') = <l,t>\&Replac'(sns,l',t')
\end{array}
$$

3.3 Very Abstract Entity Specifications

The institution of entity algebras, where "entity" stands for processes, either simple or structured (i.e. several processes interacting together), see [8],

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

provides a formal framework for the process specification.

Here, for lack of room, we cannot give full details and the motivations of $\mathcal{E}$. From a formal point of view $\mathcal{E}$ is a "subinstitution" of $\mathcal{FOE}$, in the sense that:

- **ESign** is a subcategory of **FOESign**,
- $ESen$ is the restriction of $FOESen$ to **ESign**,
- for all $E\Sigma \in |\mathbf{ESign}|$, $EMod(E\Sigma)$ is a subcategory of $FOEMod(E\Sigma)$, whose objects are called *entity algebras* and
  $EMod(\phi: E\Sigma_1 \to E\Sigma_2) = FOEMod(\phi: E\Sigma_1 \to E\Sigma_2)$,
- $EA \models^E \theta \Leftrightarrow EA \models^{FOE} \theta$.

The concurrent structure of the entities modelled by an $E\Sigma$-entity algebra (i.e. which are the process components and how are assembled) is determined by some of the operations of $E\Sigma$; thus very abstract specifications (built on the entity institution) describe classes of entity algebras on possibly different signatures, i.e. processes with possibly different concurrent structures, satisfying common properties.

$\mathcal{VE}$ is given by using the two operations ABSTRACT and EXTEND as follows (see [1] for a full definition).

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

where **EAmor** includes the elements of **YAMor** which are also morphisms in **ESign**; `esig`, `eamor`, `emor` are the restrictions of `ysig`, `yamor`, `ymor` to **ESign** and **EAmor** (in [8] it is shown that such restrictions are well-defined, i.e. they return signatures and morphisms in **ESign**).

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

where **VESign** is the subcategory of **ESign** s.t. it has the same objects and if in **VESign** there exists a morphism from $E\Sigma_1$ into $E\Sigma_2$, then the $E\Sigma_1$- and $E\Sigma_2$-entity algebras describe systems with similar concurrent structure. $COMPS$ adds to an entity signature some predicates for testing which are the subcomponents of the entities (as *Is_Sub_Entity* in the following example) and *Comps* is defined accordingly.

Example 2. 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
  $\nexists es', l \,.\, es \xrightarrow{l} es' \supset$
    – each of its subcomponents cannot perform any activity
    $\forall ec \,.\, (ec \; Is\_Sub\_Entity \; es \supset \nexists ec', l' \,.\, ec \xrightarrow{l'} ec')$

e_1 *Is_Sub_Entity* e_2 holds whenever e_1 is a subcomponent of e_2. □

Conclusions and further work

We have presented two operations on institutions, that allow to build institutions for very abstract specifications. From the practice of formal methods for software specification, it is easy to intuit that several other operations are needed in order to get meta-framework where to be able to modularly build formalisms; some more operations are presented in [6], but these are case studies rather than an organic presentation of a reasonable set of operations.

Another relevant point that we do not face here is the study of the properties of the ABSTRACT and EXTEND operations and in particular their relationships with the various notions of arrows between institutions (like horizontal/vertical composition).

References

1. E. Astesiano and G. Reggio. Entity institutions: Frameworks for dynamic systems, 1993. inpreparation.
2. E. Astesiano and G. Reggio. A metalanguage for the formal requirement specification of reactive systems. In *Proc. of FME'93*, number 670 in L.N.C.S., Berlin, 1993. Springer-Verlag.
3. C. Beierle and A. Voss. Viewing implementations as an institution. In *Proc. of Category Theory and Computer Science*, number 283 in L.N.C.S., Berlin, 1987. Springer-Verlag.
4. R. Burstall and J. Goguen. Introducing institutions. In *Logics of Programming Workshop*, number 164 in L.N.C.S. Springer Verlag, Berlin, 1984.
5. R. Burstall and J. Goguen. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1), 1992.
6. M. Cerioli and G. Reggio. Algebraic-oriented institutions. In *Proc. AMAST'93*, Workshops in Computing. Springer Verlag, Berlin, 1993.
7. P. Pepper. Transforming algebraic specifications. In *Proc. AMAST'91*, Workshops in Computing. Springer Verlag, Berlin, 1992.
8. G. Reggio. Entities: an institution for dynamic systems. In *Recent Trends in Data Type Specification*, number 534 in L.N.C.S. Springer Verlag, Berlin, 1991.
9. G. Reggio. Event logic for specifying abstract dynamic data types. In *Recent Trends in Data Type Specification*, number 655 in L.N.C.S. Springer Verlag, Berlin, 1993.
10. G. Reggio, A. Morgavi, and V. Filippi. Specification of a high-voltage substation. Technical Report PDISI-92-12, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1992.
11. G. Reichwein and J. Fiadeiro. A semantic framework for interoperability. Technical report, Departemento de matematica, Instituto superior técnico, Universidade técnica de Lisboa, 1992.
12. D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76, 1988.

This article was processed using the LaTeX macro package with LLNCS style