

# Type class polymorphism in an institutional framework

Lutz Schröder, Till Mossakowski, and Christoph Lüth

BISS, Department of Computer Science, University of Bremen

**Abstract.** Higher-order logic with shallow type class polymorphism is widely used as a specification formalism. Its polymorphic entities (types, operators, axioms) can easily be equipped with a ‘naive’ semantics defined in terms of collections of instances. However, this semantics has the unpleasant property that while model reduction preserves satisfaction of sentences, model expansion generally does not. In other words, unless further measures are taken, type class polymorphism fails to constitute a proper institution, being only a so-called rps preinstitution; this is unfortunate, as it means that one cannot use institution-independent or heterogeneous structuring languages, proof calculi, and tools with it.

Here, we suggest to remedy this problem by modifying the notion of model to include information also about its potential future extensions. Our construction works at a high level of generality in the sense that it provides, for any preinstitution, an institution in which the original preinstitution can be represented. The semantics of polymorphism used in the specification language HASCASL makes use of this result. In fact, HASCASL’s polymorphism is a special case of a general notion of polymorphism in institutions introduced here, and our construction leads to the right notion of semantic consequence when applied to this generic polymorphism. The appropriateness of the construction for other frameworks that share the same problem depends on methodological questions to be decided case by case. In particular, it turns out that our method is apparently unsuitable for observational logics, while it works well with abstract state machine formalisms such as state-based CASL.

## Introduction

The idea that a *logic* is something that comes with signatures, models, sentences and a satisfaction relation is formalized in the notion of *institution* as introduced in [15]. In practice, this concept is exploited to support genericity and heterogeneity in specification frameworks. For example, the semantics and proof calculus for structured and architectural specifications in CASL [27] is generic over institutions, and heterogeneous CASL [25, 26] uses a graph of institutions for heterogeneous specification. The central condition governing the behaviour of institutions is the *satisfaction condition*, stating that satisfaction of sentences is preserved under both model expansion and reduction.

Type class polymorphism has been used in programming languages like Haskell [31], as well as in the higher-order logic of Isabelle [38]. It is one of the central features of the recently developed specification language HASCASL [35, 36]. Little attention has been paid in the literature to the question whether type class polymorphism can be formalized as an institution, the main problem here being that with the ‘naive’ semantics, the satisfaction condition fails in the sense that satisfaction of polymorphic axioms is preserved only by model reduction, not by model expansion, because expanded models may have more types. Thus, the naive semantics defines only a so-called rps preinstitution [32] rather than an institution.

The work of [28] is an initial attempt to define an institution for polymorphism but imposes severe restrictions on signature morphisms by simply ruling out the introduction of new types. For the case of polymorphism without type classes, one solution is to parametrize the notion of model by a fixed universe of types [7, 19]; this solution, however, does not seem to be suitable for type class polymorphism.

The main goal of the present work is to provide a semantics that avoids both problems, i.e. caters for type classes and works with the usual structured specification style where the signature is built up successively. In particular, we wish to avoid restrictions on *signature morphisms*; instead, we argue that the failure of the satisfaction condition points to a flaw in the notion of *model*. The key idea is to notice that polymorphic axioms are intended as statements about all types including those yet to be declared, and that therefore models should take into account future extensions. Starting from this observation, we obtain a general procedure that transforms a preinstitution into an institution, the so-called institution of *extended models*. This construction is employed in the semantics of HASCASL. It turns out that the notions of semantic consequence and model-expansive extension engendered by the construction agree with intuitive expectations, at least in sufficiently rich logics such as the logic of HASCASL.

More generally, HASCASL's treatment of polymorphic sentences can be subsumed under a definition of polymorphic formulae in institutions introduced here. Such generic polymorphic frameworks are perfect candidates for the extended model construction, and indeed it turns out that the notion of semantic consequence in the institution of extended models over a generic polymorphic framework is simpler and more natural than the original notion.

There are several other known examples of logical frameworks where the satisfaction condition fails unless restrictions are imposed. E.g. in observational logics, signature morphisms are usually not allowed to introduce new observers [5, 16], precisely in order to rescue the satisfaction condition. Moreover, in the (non-)institution of SB-CASL [3, 4], the satisfaction condition fails for signature morphisms that introduce additional state components [3]. We discuss both these examples from a methodological perspective; it turns out that our construction cannot be recommended for the observational case, since it suppresses coinduction, while the semantics obtained for SB-CASL arguably provides the 'right' notion of semantic consequence.

The material is organized as follows. Sections 1 and 2 provide preliminary material concerning the institution-theoretic background and type class polymorphism in HASCASL. The failure of the satisfaction condition in the various settings mentioned above is treated in detail in Section 3. Section 4 defines polymorphic formulae over an institution. The construction of an institution from a given preinstitution is introduced in Section 5, and applied to generic polymorphic frameworks in Section 6. The issue of model-expansive extensions (also referred to as model-theoretically conservative extensions or, e.g. in the semantics of CASL, just as conservative extensions) is discussed in Section 7. Section 8 provides some observations on how the generic mechanism instantiates in frameworks other than type class polymorphism.

## 1 Institutions

A specification formalism is usually based on some notion of signature, model, sentence and satisfaction. These are the ingredients of the notion of *institution* as introduced by Goguen and Burstall [15]. Contrary to Barwise's notion of abstract model theory [2], the theory of institutions does not assume that signatures are algebraic signatures; indeed, nothing at all is said about signatures except that they form a class and that there are *signature morphisms*, which can be composed in some way. This amounts to stating that signatures form a *category*.

There is also nothing special assumed about the form of the *sentences* and *models*. Given a signature  $\Sigma$ , the  $\Sigma$ -sentences form just a set, while the  $\Sigma$ -models form a category (taking into account that there may be *model morphisms*). Signature morphisms lead to *translations* of sentences and of models (thus, the assignments of sentences and of models to signatures are functors). There is a contravariance between the sentence and model translations: sentences are translated *along* signature morphisms, while models are translated *against* signature morphisms.

Following [15], this is formalized as follows.

**Definition 1.** An *institution*  $I = (\mathbf{Sign}^I, \mathbf{Sen}^I, \mathbf{Mod}^I, \models^I)$  consists of

- a category  $\mathbf{Sign}^I$  of *signatures*;
- a functor  $\mathbf{Sen}^I : \mathbf{Sign}^I \rightarrow \mathbf{Set}$  giving, for each signature  $\Sigma$ , the set of *sentences*  $\mathbf{Sen}^I(\Sigma)$ , and for each signature morphism  $\sigma : \Sigma \rightarrow \Sigma'$ , the *sentence translation map*  $\mathbf{Sen}^I(\sigma) : \mathbf{Sen}^I(\Sigma) \rightarrow \mathbf{Sen}^I(\Sigma')$ , where  $\mathbf{Sen}^I(\sigma)(\varphi)$  is often written as  $\sigma\varphi$ ;
- a functor  $\mathbf{Mod}^I : (\mathbf{Sign}^I)^{op} \rightarrow \mathbf{CAT}$  (where  $\mathbf{CAT}$  denotes the quasicategory of categories and functors [1]) giving, for each signature  $\Sigma$ , the category of *models*  $\mathbf{Mod}^I(\Sigma)$ , and for each signature morphism  $\sigma : \Sigma \rightarrow \Sigma'$ , the *reduct functor*  $\mathbf{Mod}^I(\sigma) : \mathbf{Mod}^I(\Sigma') \rightarrow \mathbf{Mod}^I(\Sigma)$ , where  $\mathbf{Mod}^I(\sigma)(M')$ , the  $\sigma$ -*reduct* of  $M'$ , is often written as  $M'|_\sigma$ ; and
- a satisfaction relation  $\models_\Sigma^I \subseteq |\mathbf{Mod}^I(\Sigma)| \times \mathbf{Sen}^I(\Sigma)$  for each  $\Sigma \in \mathbf{Sign}^I$ ,

such that for each  $\sigma : \Sigma \rightarrow \Sigma'$  in  $\mathbf{Sign}^I$ , the *satisfaction condition*

$$M' \models_{\Sigma'}^I \sigma\varphi \Leftrightarrow M'|_\sigma \models_\Sigma^I \varphi$$

holds for all  $M' \in \mathbf{Mod}^I(\Sigma')$  and all  $\varphi \in \mathbf{Sen}^I(\Sigma)$ .

The notion of institutions owes much of its importance to the fact that several languages for modularizing specifications are generic over an underlying institution [11, 12, 13, 18, 27, 33]. Furthermore, institutions form the basis of *heterogeneous* frameworks such as heterogeneous CASL [25, 26]. Such frameworks require a means of interrelating institutions, i.e. some notion of morphism between institutions. There are various such notions in the literature; one of the most important ones are institution comorphisms, which essentially express that fact that one institution is *encoded* into another.

**Definition 2.** Given institutions  $I$  and  $J$ , an *institution comorphism* [17] (also called a *plain map of institutions* [21])  $\mu = (\Phi, \alpha, \beta) : I \rightarrow J$  consists of

- a functor  $\Phi : \mathbf{Sign}^I \rightarrow \mathbf{Sign}^J$ ,
- a natural transformation  $\alpha : \mathbf{Sen}^I \rightarrow \mathbf{Sen}^J \circ \Phi$ ,
- a natural transformation  $\beta : \mathbf{Mod}^J \circ \Phi^{op} \rightarrow \mathbf{Mod}^I$

such that the following *satisfaction condition* is satisfied for all  $\Sigma \in \mathbf{Sign}^I$ ,  $M' \in \mathbf{Mod}^J(\Phi(\Sigma))$  and  $\varphi \in \mathbf{Sen}^I(\Sigma)$ :

$$M' \models_{\Phi(\Sigma)}^J \alpha_\Sigma \varphi \Leftrightarrow \beta_\Sigma M' \models_\Sigma^I \varphi.$$

**Example 3.** Equational logic and first-order logic can be formalized as institutions [15], and the obvious inclusion is a comorphism.

## 2 Polymorphism in HASCASL

HASCASL is a wide-spectrum language which provides a common framework for algebraic specification and functional programming, oriented in particular towards Haskell. This is achieved by extending the algebraic specification language CASL [6]

with higher-order functions in the style of Moggi’s partial  $\lambda$ -calculus [23], type constructors, type classes, and constructor classes (for details, see [35, 36]); general recursion is specified on top of this in the style of HOLCF. The semantics of a HASCASL specification is the class of its (set-theoretic) *intensional Henkin models*: function types are interpreted by sets which need not contain all set-theoretic functions, and two functions that yield the same value on every input need not be equal.

The main point of interest for the purposes of this paper is the semantics of HASCASL’s type class oriented shallow polymorphism. A type class in HASCASL (for the sake of simplicity, we omit constructor classes here) gives rise to a subset of the *syntactical* set of types, where types are generated from basic types and type constructors, the latter either user-declared or, like function types, built-in. The set of types associated to a class is determined by the explicitly declared instances of the class. Instances may be monomorphic or polymorphic. E.g., the specification

```

class  Ord < Eq
vars  a : Type, b : Eq
types Nat : Ord;
        List a;
        List b : Eq

```

declares a class *Eq* with a subclass *Ord*, a unary type constructor *List*, and a type *Nat* (without *defining* any of these items); moreover, *Nat* is declared to be of class *Ord*, hence also of class *Eq*, and *List* is declared to produce types of class *Eq* when applied to arguments of class *Eq*. In the signature determined by the above declarations, the classes *Eq* and *Ord* coincide, both consisting precisely of the types of the form *List (List (... List Nat))*. When further instances are declared later on in the specification process, the two classes will in general be different.

Axioms and operators may be polymorphic over classes. E.g., we can write (continuing the above specification)

```

var   c : Ord
op     $-- \leq --$  : Pred(c  $\times$  c)
var   x, y, z : c
        •  $x \leq x$ 
        •  $(x \leq y \wedge y \leq z) \Rightarrow x \leq z$ 

```

This means that  $-- \leq --$  is a polymorphic predicate over class *Ord* satisfying reflexivity and transitivity. Operators and axioms may be explicitly tied to a class by means of a bracket notation, thus making up the *interface* of the class which generates proof obligations (which, like the proof obligations associated to CASL’s semantic annotations, lie outside the scope of the semantics proper) for later instantiations.

In general, polymorphic types, operators, and axioms are semantically coded out by collections of instances. That is, the effect of a polymorphic type is essentially just its contribution to the syntactic type universe; a polymorphic operator is interpreted as a family of operators, one for each instantiation of its type arguments; and a polymorphic axiom is understood as a collection of axioms, indexed over all types in the classes named in the quantifiers. This constitutes the *first level* of the semantics of polymorphism used in HASCASL; as will be explained in detail in the next section, one does not obtain an institution at this level. This deficiency is repaired at the *second level* of the semantics; this second level and the general construction behind it are the subject of this paper. The semantics of polymorphic formulas at the first level will moreover be identified as a special case of a general definition of polymorphism in institutions in Section 4.

### 3 Failures of the Satisfaction Condition

There are various features in modern specification languages that tend to cause the satisfaction condition (cf. Section 1) to fail; besides polymorphism as discussed in the previous section, this includes observational satisfaction and dynamic equations between programs in states-as-algebras frameworks such as SB-CASL [4]. Briefly, the reasons for the failures are as follows:

- **Parametric polymorphism:** if a signature morphism  $\sigma$  introduces additional types, then the translation of a polymorphic axiom  $\varphi$  may fail in a model  $M$  although  $\varphi$  holds in the reduct of  $M$  along  $\sigma$ , namely if  $\varphi$  holds for the ‘old’ types, but not for the newly introduced ones.
- **Observational equality:** if a signature morphism  $\sigma$  introduces additional observers, then observational equalities that hold in the reduct of a model  $M$  under  $\sigma$  may fail in  $M$ , since the new observers may detect previously unobservable differences.
- **dynamic equations:** if a signature morphism  $\sigma$  introduces additional state components (i.e. dynamic functions, predicates, or sorts), then dynamic equations  $p = q$  between stateful program expressions [4] that hold in the reduct  $M|_\sigma$  of a model  $M$  may fail to hold in  $M$ , since the interpretations of  $p$  and  $q$  may differ on the new state components [3].

In all these cases, only one direction of the satisfaction condition holds, so that logics with these features constitute proper rps preinstitutions; we explicitly repeat the definition [32]:

**Definition 4.** A *preinstitution* consists of a signature category equipped with model and sentence functors and a satisfaction relation in the same sense as an institution (cf. Section 1); these data are not, however, required to obey the satisfaction condition. A preinstitution is called an *rps preinstitution* (‘reducts preserve satisfaction’) if

$$M \models \sigma\varphi \quad \text{implies} \quad M|_\sigma \models \varphi$$

for all  $M, \sigma, \varphi$ , and an *eps preinstitution* (‘extensions preserve satisfaction’) if the reverse implication holds.

Let  $PI_1, PI_2$  be preinstitutions. A *preinstitution comorphism* [24]  $\mu : PI_1 \rightarrow PI_2$  consists of the same data  $(\Phi, \alpha, \beta)$  as an institution comorphism (in particular, sentence translation is covariant and model translation is contravariant), without however being required to obey the satisfaction condition as in Definition 2. A preinstitution comorphism  $\mu$  is called *rps* if

$$M \models \alpha\varphi \quad \text{implies} \quad \beta M \models \varphi,$$

and *weakly eps* if a model  $M$  satisfies  $\alpha\varphi$  whenever  $\beta K \models \sigma\varphi$  for all  $K, \sigma$  such that  $K|_{\Phi\sigma} = M$ .

Thus, an institution is a preinstitution that is simultaneously rps and eps, and a preinstitution comorphism between two institutions is an institution comorphism iff it is rps and weakly eps.

The typical remedy used hitherto to obtain institutions in the presence of the mentioned features is to restrict signature morphisms to cases where the full satisfaction condition holds. We discuss this point in more detail in Section 8; here, we just note that this is not an acceptable solution for the case of polymorphism: one has to require that signature morphisms do not introduce additional types, a restriction that effectively prevents the use of structured specifications. We emphasize that this problem is *not* solved by treating quantified types as first-class types

(higher rank polymorphism), even if one manages to work around the obstacle that the latter is inconsistent with higher order logic [10]: e.g., the restriction that signature morphisms be surjective on types is imposed also in [28], where it is needed in order to ensure preservation of coherent families of domains in a semantics of higher rank polymorphism in the style of Reynolds. In other words, ensuring coherence of polymorphic operators model-theoretically is not a feasible option.

For plain shallow polymorphism without type classes, a further alternative is to interpret the range of quantification over type variables in a fixed universe of types, i.e. some collection of sets closed under a number of constructions, rather than in the syntactical universe of declared types. This is the approach taken e.g. in [7, 19]; it is not apparently suitable for HASCASL and similar frameworks for two reasons:

- in connection with a Henkin style semantics of function types, it is unclear what closure of the type universe under function types means;
- the type universe does not give an indication of what the interpretation of type classes should be, in particular since type classes on the one hand can be entirely loose and on the other hand are meant to contain only explicitly declared instances rather than, say, all structures matching the interface.

Independently of these specific issues, a further general disadvantage of the universe approach is that the choice of a universe unduly influences semantic consequence — the particularities of the chosen universe may induce unintended semantic consequences in a rather unpredictable way, thus introducing an unnecessary degree of incompleteness of deduction. The solution chosen in the semantics of HASCASL is therefore to add a second level to the model semantics according to the general construction described below.

## 4 Generic Polymorphism

We now introduce a general notion of syntactic polymorphism in an institution which covers HASCASL’s type class polymorphism as a special case. This construction provides a wide range of examples of rps preinstitutions. We will return to this example in Section 6, where we will show that the notion of semantic consequence between polymorphic formulae induced by our generic construction of institutions from preinstitutions is not only in accordance with intuitive expectations, but also greatly simplifies the original notion.

Our construction of polymorphic formulae is similar in spirit to the *open formulae* introduced in [37]: given a signature  $\Sigma_1$ , an open  $\Sigma_1$ -formula is just a sentence  $\phi$  in some extension  $\Sigma_2$  of  $\Sigma_1$ , and a  $\Sigma_1$ -model  $M$  satisfies such a formula if all its expansions to  $\Sigma_2$  satisfy  $\phi$ . In typical algebraic settings, this produces exactly the right kind of first or higher order quantification if  $\Sigma_2$  introduces only additional constants or function symbols, respectively; essentially, the new symbols then play the role of universally quantified variables. However, the given notion of satisfaction is rather too strong if  $\Sigma_2$  introduces additional types; since new sorts and function symbols involving new sorts (including instances of polymorphic operators for new sorts) can be interpreted with arbitrary malevolence in extensions of  $M$ , most open formulae involving such a  $\Sigma_2$  will in fact be unsatisfiable.

Thus, we need a relaxed notion of satisfaction in order to arrive at the right notion of universal quantification over types. The idea is to require satisfaction of  $\phi$  as above not for *all* extensions of  $M$ , but only for *extensions by syntactic definition*, i.e. the new signature items in  $\Sigma_2$  have to be interpreted in terms of the base signature  $\Sigma_1$ . Of course, the involved notion of interpretation will have to be sufficiently general. E.g., we will want to interpret function symbols by terms, type constants by composite types etc. — in other words, we will need to use derived signature morphisms. All this is formalized as follows.

**Definition 5.** An *institution with signature variables* is an institution  $I$  with a distinguished object-full subcategory  $\mathbf{Var}$  of the signature category  $\mathbf{Sign}$  (i.e.  $\mathbf{Var}$  need not be full in  $\mathbf{Sign}$ , but contains all objects of  $\mathbf{Sign}$ ) whose morphisms are called *signature variables*. Signature variables are assumed to be *pushout-stable*, i.e. pushouts of signature variables along  $\mathbf{Sign}$ -morphisms exist and are signature variables. (Morphisms in  $\mathbf{Sign}$  should be thought of as derived signature morphisms.)

In  $I$ , a *polymorphic formula*  $\forall\sigma.\phi$  over a signature  $\Sigma_1$  consists of a signature variable  $\sigma : \Sigma_1 \rightarrow \Sigma_2$  and a  $\Sigma_2$ -sentence  $\phi$ . A  $\Sigma_1$ -model  $M$  *satisfies*  $\forall\sigma.\phi$  if

$$M \models \tau\phi \quad \text{for all } \tau \text{ in } \mathbf{Sign} \text{ such that } \tau \circ \sigma = id.$$

A sentence  $\tau\phi$  as above is called an *instance* of  $\forall\sigma.\phi$ . The *translation*  $\rho(\forall\sigma.\phi)$  of  $\forall\sigma.\phi$  along a signature morphism  $\rho : \Sigma_1 \rightarrow \Sigma_3$  is defined to be  $\forall\bar{\sigma}.\bar{\rho}\phi$ , where

$$\begin{array}{ccc} \Sigma_2 & \xrightarrow{\bar{\rho}} & \bullet \\ \sigma \uparrow & & \uparrow \bar{\sigma} \\ \Sigma_1 & \xrightarrow{\rho} & \Sigma_3 \end{array}$$

is a pushout; note that  $\bar{\sigma}$  is indeed a signature variable. (This definition determines the translation only up to isomorphism; for similar reasons as given in Remark 5.1. of [37], this is not actually a problem.)

The *polymorphic preinstitution*  $\text{Poly}(I)$  over  $I$  is given as follows: the notions of signature, model, and model reduction are inherited from  $I$ ;  $\Sigma$ -sentences are polymorphic formulae over  $\Sigma$ ; satisfaction and sentence translation are as above.

The sentences of  $I$  can be coded in  $\text{Poly}(I)$ : a  $\Sigma$ -sentence  $\phi$  in  $I$  is equivalent to the polymorphic formula  $\forall id_\Sigma.\phi$ , where  $id_\Sigma$  is indeed a signature variable thanks to object-fullness of  $\mathbf{Var}$  in  $\mathbf{Sign}$ .

**Example 6.** An example of the syntactic polymorphism described above is HASCASL's type class polymorphism. The base institution is essentially as in the first level of the HASCASL semantics, except that polymorphic sentences are excluded, so that we actually obtain an institution rather than an rps preinstitution. (Note that the base institution does have polymorphic types and operators. In particular, signature morphisms can translate polymorphic operators only as a whole, not instance by instance.) Signature morphisms are, as announced above, *derived signature morphisms* which map

- operator constants to terms;
- type constructors to  $\lambda$ -expressions which denote composite type constructors possibly containing subtype formation; and
- classes to subsets of the syntactic type universe.

A signature variable in this institution is an injective *plain* HASCASL signature morphism (which maps types to types, operators to operators etc. as usual) which is bijective on all syntactic entities except types. (This illustrates the necessity of the restricted cocompleteness requirement for institutions with signature variables: pushouts of derived signature morphisms in general fail to exist, while pushouts of derived signature morphisms along signature variables do exist; this phenomenon is typical of derived signature morphisms in general.) Then, polymorphic formulae and their satisfaction as defined above coincide with the corresponding notions in HASCASL as explained in Section 2. E.g., if  $\sigma : \Sigma_1 \hookrightarrow \Sigma_2$  extends  $\Sigma_1$  by a single new type constant  $a$ , then the polymorphic formula  $\forall\sigma.\phi$  is equivalent to the polymorphic HASCASL sentence  $\forall a : \text{Type}.\phi$ : the left inverses  $\tau$  of  $\sigma$  correspond to

the possible instantiations of the type variable  $a$  in  $\Sigma_1$ . Note that the interpretation of instances of polymorphic operators involving  $a$  is forced by the interpretation of  $a$ , since, as emphasized above, signature morphisms map polymorphic operators as single entities.

By the above example and Section 3, it is clear that the polymorphic preinstitution  $\text{Poly}(I)$  will in general fail to be an institution. However, we have

**Theorem 7.** *The polymorphic preinstitution  $\text{Poly}(I)$  is an rps preinstitution.*

## 5 A Generic Institutionalization

We now describe a general process that transforms preinstitutions into institutions. We begin with a heuristic observation regarding the intended meaning of polymorphic definitions. Consider the specification

```
spec COMPOSITION =
  vars  a, b, c : Type
  op    comp : (b → c) → (a → b) → a → c
  vars  f : b → c; g : a → b
        • comp f g = λx : a . f (g x)
```

where for the sake of the argument we abuse HASCASL as a notation for the simply typed  $\lambda$ -calculus with shallow polymorphism in much the same sense as described in Section 2, the only real point of this being the assumption that, unlike in actual HASCASL, there is no unit type. On the first level of the semantics as described in Section 2, COMPOSITION is model-theoretically entirely vacuous, since the syntactic set of types is empty and hence the polymorphic axiom is trivially satisfied in ‘all’ models of the signature (there is in fact only one model, since the signature is effectively empty). This is clearly not the intention of COMPOSITION. Indeed this specification is necessarily meant as a building block for other specifications that import the polymorphic operator and its definition, which then induce instances according to the ambient signature. In other words, the real purpose of COMPOSITION is apparently to say something about the interpretation of *comp* at *all* types, even those not yet declared. Thus, a model of the specification should contain information not only about the interpretation of the presently declared signature, but also about all ‘future’ extensions of this interpretation. This is the motivation for the following definitions:

**Definition 8.** Let  $PI$  be a preinstitution. An *extended model* of a signature  $\Sigma_1$  is a pair  $(N, \sigma)$ , where  $\sigma : \Sigma_1 \rightarrow \Sigma_2$  is a signature morphism and  $N$  is a  $\Sigma_2$ -model in  $PI$ . The *reduct*  $(N, \sigma)|_\tau$  of  $(N, \sigma)$  along a signature morphism  $\tau$  is  $(N, \sigma \circ \tau)$ . The extended model  $(N, \sigma)$  *satisfies* a sentence  $\varphi$  if

$$N \models \sigma\varphi$$

in  $PI$ .

We record explicitly

**Theorem and Definition 9.** *The extended models, together with the original notions of signature and sentence from  $PI$ , form an institution, called the institution of extended models and denoted  $\text{Ext}(PI)$ .*

*Proof.* Functoriality of reduction is easy to see. To check the satisfaction condition, let  $\tau : \Sigma_1 \rightarrow \Sigma_2$  be a signature morphism, let  $\varphi$  be a  $\Sigma_1$ -sentence, and let  $(N, \sigma)$  be an extended  $\Sigma_2$ -model. Then  $(N, \sigma) \models \tau\varphi$  in  $\text{Ext}(PI)$  iff  $N \models \sigma\tau\varphi$  in  $PI$  iff  $(N, \sigma)|_\tau = (N, \sigma \circ \tau)$  satisfies  $\varphi$  in  $\text{Ext}(PI)$ .  $\square$

The semantic consequence relation in  $\text{Ext}(PI)$  is precisely as expected:

**Proposition 10.** *A  $\Sigma_1$ -sentence  $\psi$  is a semantic consequence of a set  $\Phi$  of  $\Sigma_1$ -sentences in  $\text{Ext}(PI)$  iff*

$$\sigma\Phi \models \sigma\psi$$

*in  $PI$  for each signature morphism  $\sigma : \Sigma_1 \rightarrow \Sigma_2$ .*

*Proof.* ‘If’: trivial.

‘Only if’: let  $\sigma : \Sigma_1 \rightarrow \Sigma_2$  be a signature morphism, and let  $N$  be a  $\Sigma_2$ -model such that  $N \models \sigma\Phi$  in  $PI$ . Then the extended model  $(N, \sigma)$  satisfies  $\Phi$  and hence also  $\psi$ , i.e. we have  $N \models \sigma\psi$ .  $\square$

That is, a formula is a semantic consequence of a specification  $Sp = (\Sigma, \Phi)$  (where  $\Phi$  is a set of  $\Sigma$ -sentences) iff this is the case, in  $PI$ , in all extensions of  $Sp$ .

**Example 11.** In the example specification COMPOSITION from Section 3, all formulae are semantic consequences on the first level, i.e. in  $PI$ , since all formulae are vacuously true. This pathology disappears in  $\text{Ext}(PI)$ , where semantic consequences of the specification are only those formulae that follow from the definition of composition independently of how many types are introduced, such as e.g. associativity of composition. Thus, the notion of semantic consequence at the second level, unlike the one at the first level, conforms to intuitive expectations. We will make this more precise in Section 6.

One can give a concise description of extensions in  $\text{Ext}(PI)$ :

**Lemma 12.** *The extensions of an extended model  $(N, \tau)$  along a signature morphism  $\sigma$  are precisely the extended models  $(N, \rho)$  where  $\tau = \rho \circ \sigma$ .*

We can represent  $PI$  in  $\text{Ext}(PI)$  by a preinstitution comorphism (cf. Definition 4)

$$\eta : PI \rightarrow \text{Ext}(PI)$$

which is the identity on signatures and sentences, and takes every extended model to its base model.

**Proposition 13.** *The comorphism  $\eta$  is weakly eps. Moreover,  $\eta$  is rps if  $PI$  is rps.*

**Remark 14.** Interestingly, the concept of extended model is close to the *very abstract* or *hyper-loose semantics* as introduced in [9, 30], where models may interpret more symbols than just the ones named in their signature. This is used e.g. in the semantics of RSL [14].

There are two crucial differences here. The first is of motivational nature: the purpose of very abstract semantics is to ensure that refinement is model class inclusion; there is no intended connection with repairing the satisfaction condition, and in fact, the construction described in [9] is explicitly intended as a construction *on institutions* (one of the example applications given in [9, 30] is to the institution of many-sorted first order logic). Note that, when applied to institutions, the very abstract semantics is equivalent to the original semantics in terms of the engendered semantic consequence relation.

Secondly, at a more technical level, the phrase ‘models may interpret additional symbols’ means that very abstract semantics limits the notion of model to extended models with injective signature morphisms; the main technical content of [9] is to solve the difficulties caused by this restriction w.r.t. model reduction. For the purposes pursued here, the restriction to injective extensions is not only unnecessary, but would indeed invalidate our main result; i.e. for models of polymorphism modeled along the construction of [9], the satisfaction condition would still fail.

Taking  $PI$  as the first level of the HASCASL semantics (cf. Section 2), we define the *second level* of the semantics [36] to be given by  $\text{Ext}(PI)$ .

## 6 Semantic Consequence for Generic Polymorphism

We now investigate the implications of the extended model construction explained in Section 5 in relation to the generic polymorphism introduced in Section 4 — recall that generic polymorphism in general leads only to an rps preinstitution. For the remainder of this section, let  $I$  be an institution with signature variables, and let  $\text{Poly}(I)$  denote the polymorphic preinstitution over  $I$  as defined in Section 4.

Let  $\forall\sigma.\phi$  and  $\forall\rho.\psi$  be polymorphic formulae over a signature  $\Sigma_1$ . It is easy to check that  $\forall\rho.\psi$  is a semantic consequence of  $\forall\sigma.\phi$  in  $\text{Poly}(I)$  iff

$$\{\tau\phi \mid \tau \circ \sigma = id\} \models \pi\psi$$

in  $I$  for each signature morphism  $\pi$  such that  $\pi \circ \sigma = id$ . This is rather unpleasant, since it means we have to prove a possibly infinite number of semantic consequences, one for each instance  $\pi\psi$  of  $\forall\rho.\psi$  in  $\Sigma_1$ . Fortunately, the (stronger) notion of semantic consequence in the institution  $\text{Ext}(\text{Poly}(I))$  is much more tractable:

**Theorem 15.** *In  $\text{Ext}(\text{Poly}(I))$ ,  $\forall\rho.\psi$  is a semantic consequence of  $\forall\sigma.\phi$  iff*

$$\rho(\forall\sigma.\phi) \models \psi$$

in  $\text{Poly}(I)$  (or, since  $\psi$  enjoys eps, equivalently in  $\text{Ext}(\text{Poly}(I))$ )

(Recall that  $\rho(\forall\sigma.\phi) = \forall\bar{\sigma}.\bar{\rho}\phi$ , where  $(\bar{\rho}, \bar{\sigma})$  is the pushout of  $(\sigma, \rho)$ ). The above condition can be equivalently rephrased as the semantic consequence

$$\{\lambda\phi \mid \lambda \circ \sigma = \rho\} \models \psi \quad (*)$$

in  $I$ . Thus, unlike proofs of semantic consequence in  $\text{Poly}(I)$  as described above, proofs in  $\text{Ext}(\text{Poly}(I))$  are actually feasible, since we have to prove only a single generic instance of the goal, rather than all instances that exist in the base signature due to pure syntactic happenstance. Moreover,

*any sound and complete deduction system for  $I$  induces a sound and complete deduction system for  $\text{Ext}(\text{Poly}(I))$ ,*

while for  $\text{Poly}(I)$ , one will in general only obtain a sound but not complete deduction system.

The formulation of semantic consequence given in the theorem is exactly what one would intuitively expect: we fix the additional syntactic material quantified over by  $\rho$  and prove  $\psi$  only for this fixed instance; in the proof, we are allowed to make use of all instances of  $\phi$ , including instances involving the new syntactic material. Proofs of polymorphic formulas e.g. in Isabelle [29] work in precisely this way, which we have now provided with a semantic foundation.

*Proof (Theorem 15).* ‘Only If’: by Proposition 10, we have  $\rho(\forall\sigma.\phi) \models \rho(\forall\rho.\psi)$ , and  $\psi$  is an instance of  $\rho(\forall\rho.\psi)$ . The latter follows from the universal property of the pushout of  $\rho$  with itself.

‘If’: let  $\Sigma_1$  be the base signature of  $\forall\sigma.\phi$  and  $\forall\rho.\psi$ , let  $\kappa : \Sigma_1 \rightarrow \Sigma_2$  be a signature morphism, and let

$$\begin{array}{ccc} \bullet & \xrightarrow{\bar{\kappa}_\sigma} & \bullet \\ \sigma \uparrow & & \uparrow \bar{\sigma} \\ \Sigma_1 & \xrightarrow{\kappa} & \Sigma_2 \end{array} \quad \text{and} \quad \begin{array}{ccc} \bullet & \xrightarrow{\bar{\kappa}_\rho} & \bullet \\ \rho \uparrow & & \uparrow \bar{\rho} \\ \Sigma_1 & \xrightarrow{\kappa} & \Sigma_2 \end{array}$$

be the associated pushout diagrams. Then  $\kappa(\forall\sigma.\phi) = \forall\bar{\sigma}.\bar{\kappa}_\sigma\phi$  and  $\kappa(\forall\rho.\psi) = \forall\bar{\rho}.\bar{\kappa}_\rho\psi$ . By Proposition 10, we thus have to prove

$$\forall\bar{\sigma}.\bar{\kappa}_\sigma\phi \models \forall\bar{\rho}.\bar{\kappa}_\rho\psi$$

in  $\text{Poly}(I)$ , i.e. given a model  $M$  such that  $M \models \forall\bar{\sigma}.\bar{\kappa}_\sigma\phi$  and  $\tau$  such that  $\tau\bar{\rho} = id$ , we have to show  $M \models \tau\bar{\kappa}_\rho\psi$  in  $I$ . Since semantic consequence in  $I$  is stable under translation, this reduces by (\*) above to showing  $M \models \tau\bar{\kappa}_\rho\lambda\phi$  for all  $\lambda$  such that  $\lambda \circ \sigma = \rho$ . For such a  $\lambda$ , we have  $\tau\bar{\kappa}_\rho\lambda\sigma = \tau\bar{\kappa}_\rho\rho = \tau\bar{\rho}\kappa = \kappa$ , so that the pushout property yields  $\nu$  such that  $\nu\bar{\sigma} = id$  and  $\nu\bar{\kappa}_\sigma = \tau\bar{\kappa}_\rho\lambda$ . Then  $M$  satisfies the instance  $\nu\bar{\kappa}_\sigma\phi$  of  $\forall\bar{\sigma}.\bar{\kappa}_\sigma\phi$ ; but  $\nu\bar{\kappa}_\sigma\phi = \tau\bar{\kappa}_\rho\lambda\phi$ .  $\square$

## 7 Model-Theoretic Conservativity

While the semantic consequence relation engendered by the extended model construction is without further ado precisely the ‘right’ one, the issue of model expansion, i.e. of conservativity in the model-theoretic sense as used e.g. in CASL, is somewhat more subtle. We recall a few definitions:

**Definition 16.** A *theory* in a (pre-)institution is a pair  $Sp = (\Sigma, \Phi)$  consisting of a signature  $\Sigma$  and a set  $\Phi$  of  $\Sigma$ -sentences. A *model* of  $Sp$  is a  $\Sigma$ -model  $M$  such that  $M \models \Phi$ . A theory is *consistent* if it has a model. A signature morphism  $\sigma : \Sigma_1 \rightarrow \Sigma_2$  is a *theory morphism*  $(\Sigma_1, \Phi_1) \rightarrow (\Sigma_2, \Phi_2)$  if

$$\Phi_2 \models \sigma\Phi_1.$$

A theory morphism  $\sigma : Sp_1 \rightarrow Sp_2$  is *model-theoretically conservative* or *model-expansive* if every model  $M$  of  $Sp_1$  has an  $Sp_2$ -extension, i.e. a model  $N$  of  $Sp_2$  such that  $N|_\sigma = M$ .

Notice that by Proposition 10 and Example 11, the notion of theory morphism in  $\text{Ext}(PI)$  is in general properly stronger than in  $PI$ .

**Proposition 17.** A theory is consistent in an rps preinstitution  $PI$  iff it is consistent in  $\text{Ext}(PI)$ .

Typical extensions that would be expected to be model-expansive e.g. in HAS-CASL are (recursive) function definitions, loose declarations of new signature elements, and declarations of free datatypes. An apparent obstacle to model-expansivity of such extensions at the second level of the semantics is Part (i) of the following observation:

**Proposition 18.** Let  $PI$  be an rps preinstitution, and let  $\sigma : (\Sigma_1, \Phi_1) \rightarrow (\Sigma_2, \Phi_2)$  be a theory morphism in  $\text{Ext}(PI)$ . Then the following holds:

- (i) If  $\sigma$  is model-expansive in  $\text{Ext}(PI)$  and  $(\Sigma_1, \Phi_1)$  is consistent, then  $\sigma$  is a section as a signature morphism; i.e. there exists a signature morphism  $\tau : \Sigma_2 \rightarrow \Sigma_1$  such that  $\tau \circ \sigma = id$ .
- (ii) If  $\sigma$  is a section as a theory morphism in  $\text{Ext}(PI)$ , i.e. there exists a theory morphism  $\tau : (\Sigma_2, \Phi_2) \rightarrow (\Sigma_1, \Phi_1)$  such that  $\tau \circ \sigma = id$ , then  $\sigma$  is model-expansive.

*Proof.* (i): By assumption and the rps condition,  $(\Sigma_1, \Phi_1)$  has a model  $(M, id)$  in  $\text{Ext}(PI)$ . By Lemma 12, existence of an extension of this model along  $\sigma$  implies that  $\sigma$  is a section.

(ii): Straightforward.  $\square$

When plain signature morphisms are used, which typically map type constants to type constants, operators to operators etc., then the necessary condition above is clearly too restrictive; essentially, the only model-expansive extensions one obtains are those that define symbols by other symbols already present. The solution to this is to use *derived* signature morphisms instead, which typically are allowed to map, say, type constants to composite types, operators to terms, and the like; by the sufficient condition (ii) above, one then obtains as model-expansive extensions all declarations and definitions which can be implemented by some composite object in the present theory.

In the case of HASCASL, the notion of derived signature morphism required here is the one already given in Example 6. Thanks to the richness of HASCASL specifications, the model-expansive extensions are indeed the expected ones under this definition; this includes

- equational definitions
- well-founded recursive definitions of functions into types that admit a unique description operator [34]
- general recursive definitions over cpo’s
- inductive datatype definitions, provided that the base theory already contains the natural numbers (this is a categorical result inherited from topos theory [22])
- class declarations.

In general, it depends on the expressive power of signatures and theories in the preinstitution at hand whether or not using derived signature morphisms leads to a satisfactory notion of model-expansivity. It should however be noted that there is usually quite some latitude in the definition of derived signature morphism; many forms of extensions can be made model-expansive by just giving a more liberal definition of what a derived signature morphism can do.

## 8 Application to Other Frameworks

We now briefly discuss the effects of the extended model construction in other frameworks where the satisfaction condition may fail, to wit, in observational and state-based frameworks as described in Section 3. Of course, the construction will always work in principle; however, the question remains whether or not the ensuing semantic modifications are methodologically desirable, and what the actual benefits are. Here, we will concentrate on two issues:

- A) Is the notion of semantic consequence engendered by the extended model construction the expected one? I.e., in view of Proposition 10, is semantic consequence intended to be independent of the surrounding signature?
- B) Is the alternative solution of restricting signature morphisms acceptable?

We have seen that, in the case of type class polymorphism, the answer is ‘yes’ to Question A) and ‘no’ to Question B): semantic consequences that hold only due to the particular nature of the presently declared types can be regarded as unwanted side effects, and limiting signature morphisms to be surjective on types is not an option.

The situation is different with observational satisfaction. It is precisely the point of having distinguished observable operations or sorts that these govern the notion of observational equality, and moreover that the given set of observers determines a proof principle for observational equality, namely coinduction. This proof principle is lost when extended models are considered (in a setting with unrestricted signature morphisms): since deduction then has to work within arbitrary signature extensions that may introduce any number of additional observers, the notion of

semantic consequence for extended models is just semantic consequence in standard equational logic. This is clearly not the desired effect, so that the notion of extended model cannot in fact be considered suitable for observational specification. It is thus lucky that, given this negative answer to Question A), the answer to Question B) is affirmative: it is common practice to restrict signature morphisms of observational specifications in such a way that extensions never introduce new observers [5, 16]. This forces a specification style where all observers are introduced in one go at the beginning, being regarded as constituting the requirements on the system, and the non-observable part, i.e. the implementation, is added later; indeed, this specification style is explicitly advocated e.g. in [20].

Finally, let us have a look at the specification of stateful systems in the states-as-algebras paradigm as used e.g. in the specification language SB-CASL [4]. The problem here, as pointed out in Section 3, are so-called *dynamic equations* between program-like expressions called *transition terms* in SB-CASL (besides these dynamic equations, SB-CASL also features pre- and postconditions, which are however unproblematic w.r.t. the satisfaction condition). The purpose of dynamic equations lies both in the (possibly recursive) definition of procedures and in their loose equational specification e.g. as inverses of other procedures (a very simple example of this is given in [4]). As indicated in Section 3, dynamic equations may break in model expansions when signature morphisms introduce additional state components.

The methodology of state-based specification in this sense is not as yet well developed, so that we feel entitled to pitch in our own bit of philosophy, as follows. Concerning Question B) above, it seems undesirable to have a development paradigm where the specification process starts with defining the entire state space in full detail and only then allows the formulation of requirements for programs that work on this state space; to the contrary, one would normally wish to start with the requirements, mentioning only the parts of the state space relevant for input and output, and then work out the detailed design of the state space. As to Question A), it appears for rather the same reason that semantic consequences that hold only due to an insufficiently detailed description of the state space should be regarded as spurious, so that the notion of semantic consequence induced by the extended model construction is indeed an improvement over the original one. As an extreme example, consider a specification that introduces some procedure names, but no dynamic signature components at all (presumably with the intention to specify these in later extensions), i.e. induces a trivial state space; in SB-CASL, such a specification might look as follows:

```
spec SP =
  proc p, q
  pre p: True
  pre q: True
```

(the two preconditions express that  $p$  and  $q$  terminate). Then, unless extended models are used, any two terminating programs (transition terms) would be equal, i.e. their equality is a semantic consequence of the precondition expressing their termination; in particular, the above specification implies the dynamic equation  $p = q$ . We argue that this sort of semantic consequence is actually a pathology, which is eliminated by our extended model construction.

## 9 Conclusion

Starting from the problem that type class polymorphism does not enjoy the satisfaction condition of institutions, but only the *reduction preserves satisfaction* (rps) half, we suggest a general construction of institutions from preinstitutions. The

construction is based on the idea that a model of a specification should contain information not only about the interpretation of the presently declared signature, but also about all ‘future’ extensions of this interpretation. Consequently, the *extended models* of a signature are defined to consist of a signature extension and a model of the extended signature. The arising notion of semantic consequence is the expected one, namely, semantic consequence in all signature extensions in the original preinstitution. Moreover, in sufficiently rich logics such as the HASCASL logic, one also obtains the expected model-expansive extensions.

The semantics of polymorphism used in HASCASL makes use of this result, so that HASCASL does indeed fit into the institution-independent framework of CASL. We have also investigated the use of our construction in other frameworks where the satisfaction condition fails for unrestricted signature morphisms, the result being that the implications of our constructions are methodologically undesirable in the case of observational satisfaction, but beneficial in the case of dynamic equations in a states-as-algebras framework. The suitability of our approach for security formalisms, which also exhibit the phenomenon that security assertions tend to be unstable under refinement [8], is under investigation.

A particularly pleasing point is that HASCASL’s polymorphic sentences can be subsumed under a general definition of polymorphic formulae over institutions; the extended model construction, when applied to such generic polymorphic frameworks, leads to a very natural notion of semantic consequence which agrees with proof principles used e.g. in Isabelle [29]. In this sense, our method provides a semantic basis for existing proof methods.

## Acknowledgements

The authors wish to thank Hubert Baumeister for valuable information about SB-CASL and Tom Maibaum for useful hints and discussions.

## References

- [1] J. Adámek, H. Herrlich, and G. E. Strecker, *Abstract and concrete categories*, Wiley Interscience, 1990.
- [2] J. Barwise, *Axioms for abstract model theory*, Ann. Math. Logic **7** (1974), 221–265.
- [3] H. Baumeister, *An institution for SB-CASL*, talk presented at the 15th International Workshop on Algebraic Development Techniques, Genova, 2001.
- [4] H. Baumeister and A. Zamulin, *State-based extension of CASL*, Integrated Formal Methods, LNCS, vol. 1945, Springer, 2000, pp. 3–24.
- [5] M. Bidoit and R. Hennicker, *On the integration of observability and reachability concepts*, Foundations of Software Science and Computation Structures, LNCS, vol. 2303, Springer, 2002, pp. 21–36.
- [6] M. Bidoit and P. D. Mosses, *CASL user manual*, LNCS, vol. 2900, Springer, 2004.
- [7] T. Borzyszkowski, *Higher-order logic and theorem proving for structured specifications*, Recent Trends in Algebraic Development Techniques, (WADT 99), LNCS, vol. 1827, Springer, 2000, pp. 401–418.
- [8] A. Bossi, R. Focardi, C. Piazza, and S. Rossi, *Refinement operators and information flow security*, Software Engineering and Formal Methods, IEEE Computer Society Press, 2003, pp. 44–53.
- [9] M. Cerioli and G. Reggio, *Very abstract specifications: a formalism independent approach*, Math. Struct. Comput. Sci. **8** (1998), 17–66.
- [10] T. Coquand, *An analysis of Girard’s paradox*, Logic in Computer Science, IEEE, 1986, pp. 227–236.
- [11] R. Diaconescu, J. Goguen, and P. Stefanescu, *Logical support for modularisation*, Workshop on Logical Frameworks, Programming Research Group, Oxford University, 1991.
- [12] F. Durán and J. Meseguer, *Structured theories and institutions*, Category Theory and Computer Science, ENTCS, vol. 29, 1999.

- [13] H. Ehrig and B. Mahr, *Fundamentals of algebraic specification 2*, Springer, 1990.
- [14] C. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. Bendix Nielson, S. Prehn, and K. R. Wagner, *The Raise Specification Language*, Prentice Hall, 1992.
- [15] J. Goguen and R. Burstall, *Institutions: Abstract model theory for specification and programming*, J. ACM **39** (1992), 95–146.
- [16] J. Goguen and G. Malcolm, *A hidden agenda*, Theoret. Comput. Sci. **245** (2000), 55–101.
- [17] J. Goguen and G. Rosu, *Institution morphisms*, Formal aspects of computing **13** (2002), 274–307.
- [18] J. Goguen and W. Tracz, *An implementation-oriented semantics for module composition*, Foundations of Component-Based Systems, Cambridge, 2000, pp. 231–263.
- [19] R. Kubiak, A. Borzyszkowski, and S. Sokolowski, *A set-theoretic model for a typed polymorphic lambda calculus — a contribution to MetaSoft*, VDM: The Way Ahead, LNCS, vol. 328, Springer, 1988, pp. 267–298.
- [20] A. Kurz, *Logics for coalgebras and applications to computer science*, Ph.D. thesis, Universität München, 2000.
- [21] J. Meseguer, *General logics*, Logic Colloquium 87, North Holland, 1989, pp. 275–329.
- [22] I. Moerdijk and E. Palmgren, *Wellfounded trees in categories*, Ann. Pure Appl. Logic **104** (2000), 189–218.
- [23] E. Moggi, *Categories of partial morphisms and the  $\lambda_p$ -calculus*, Category Theory and Computer Programming, LNCS, vol. 240, Springer, 1986, pp. 242–251.
- [24] T. Mossakowski, *Representations, hierarchies and graphs of institutions*, Ph.D. thesis, Universität Bremen, 1996, also: Logos-Verlag, 2001.
- [25] ———, *Foundations of heterogeneous specification*, Recent Trends in Algebraic Development Techniques (WADT 02), LNCS, vol. 2755, Springer, 2003, pp. 359–375.
- [26] ———, *HETCASL - heterogeneous specification. Language summary*, 2004, <http://www.tzi.de/cofi/hetcasl>.
- [27] P. D. Mosses (ed.), *CASL reference manual*, LNCS, vol. 2960, Springer, 2004.
- [28] M. Nielsen and U. Pletat, *Polymorphism in an institutional framework*, Tech. report, Technical University of Denmark, 1986.
- [29] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL — a proof assistant for higher-order logic*, LNCS, vol. 2283, Springer, 2002.
- [30] P. Pepper, *Transforming algebraic specifications – lessons learnt from an example*, Constructing Programs from Specifications, Elsevier, 1991, pp. 1–27.
- [31] S. Peyton-Jones (ed.), *Haskell 98 language and libraries — the revised report*, Cambridge, 2003, also: J. Funct. Programming **13** (2003).
- [32] A. Salibra and G. Scollo, *A soft stairway to institutions*, Recent Trends in Data Type Specification (WADT 91), LNCS, vol. 655, Springer, 1993, pp. 310–329.
- [33] D. Sannella and A. Tarlecki, *Specifications in an arbitrary institution*, Information and Computation **76** (1988), 165–210.
- [34] L. Schröder, *The HASCASL prologue: categorical syntax and semantics of the partial  $\lambda$ -calculus*, available as <http://www.informatik.uni-bremen.de/~lschrode/hascasl/plam.ps>
- [35] L. Schröder and T. Mossakowski, *HASCASL: Towards integrated specification and development of functional programs*, Algebraic Methodology And Software Technology, LNCS, vol. 2422, Springer, 2002, pp. 99–116.
- [36] L. Schröder, T. Mossakowski, and C. Maeder, *HASCASL – Integrated functional specification and programming. Language summary*, available under [http://www.informatik.uni-bremen.de/agbkb/forschung/formal\\_methods/CoFI/HasCASL](http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HasCASL)
- [37] A. Tarlecki, *Quasi-varieties in abstract algebraic institutions*, J. Comput. System Sci. **33** (1986), 333–360.
- [38] M. Wenzel, *Type classes and overloading in higher-order logic*, Theorem Proving in Higher Order Logics, LNCS, vol. 1275, Springer, 1997, pp. 307–322.