Math. Struct. in Comp. Science (1999), *vol.* 9, *pp.* 21–62. Printed in the United Kingdom

Abstract and behaviour module specifications

FELIX CORNELIUS[†], MICHAEL BALDAMUS[†],
HARTMUT EHRIG[†] and FERNANDO OREJAS[‡]

[†] *Berlin University of Technology*
[‡] *Universitat Politècnica de Catalunya, Barcelona*

Received 2 August 1993; revised 6 January 1998

The theory of algebraic module specifications and modular systems was developed initially mainly on the basis of equational algebraic specifications. We show that it is in fact almost independent of what kind of underlying specification framework is chosen. More specifically, we present a formulation where this framework appears as an indexed category or, equivalently, *specification frame*. The ensuing theory is called the *theory of abstract module specifications*. We are able to prove main results concerning the correctness and compositionality of abstract module specifications in a purely categorical way, assuming the existence of pushouts of morphisms between abstract specifications that allow model amalgamation, functor extension and/or suitable free constructions. Then, by instantiating the theory of abstract module specifications to the behaviour specification frame in the sense of Nivela and Orejas, we obtain a theory of behaviour module specifications.

1. Introduction

The importance of decomposing large software systems into smaller units, called modules, to improve their clarity, facilitate proofs of correctness, and support re-usability has been widely recognized within the programming and software engineering community. For all stages of the software development process, modules (module specifications) are seen as completely self-contained units that can be developed independently and then interconnected with each other. An algebraic module specification MOD as developed in Ehrig and Mahr (1985), Blum *et al.* (1987) and Ehrig and Mahr (1990) consists of four components $MOD = (PAR, EXP, IMP, BOD)$ as depicted in Figure 1.

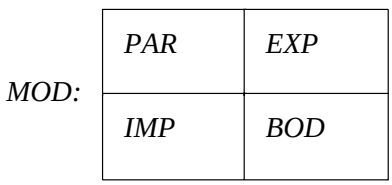


Fig. 1. Structure of an algebraic module specification

These components are equational algebraic specifications in the sense of Zilles (1974), Thatcher *et al.* (1978) and Ehrig and Mahr (1985). The export EXP and the import

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

IMP represent the interfaces of the module, and the parameter *PAR* is a part common to both import and export. It represents a part of the parameters of the whole system. Also, the three interface specifications PAR, EXP and *IMP* are allowed to be algebraic specifications with constraints. This enables us to express requirements for operations and domains in the interfaces using suitable formalisms. Finally, the body *BOD* represents the constructive part of the module. It makes use of the resources provided by the import and offers the resources provided by the export.

The *semantics of a module specification MOD* is given by the loose semantics, *i.e.*, the class of all algebras satisfying the interface specifications PAR, EXP and *IMP*, respectively, a *free construction* from import to body algebras, and a *behaviour construction* from import to export algebras given by restriction of the free construction to the export part.

A module specification is called *internally correct* if the free construction protects import algebras and if the behaviour construction transforms import algebras satisfying the import constraints into export algebras satisfying the export constraints.

Basic interconnection mechanisms applied to module specifications are: *composition*, *union* and *actualization*. The composition operation is the most important. This is sketched in Figure 2.

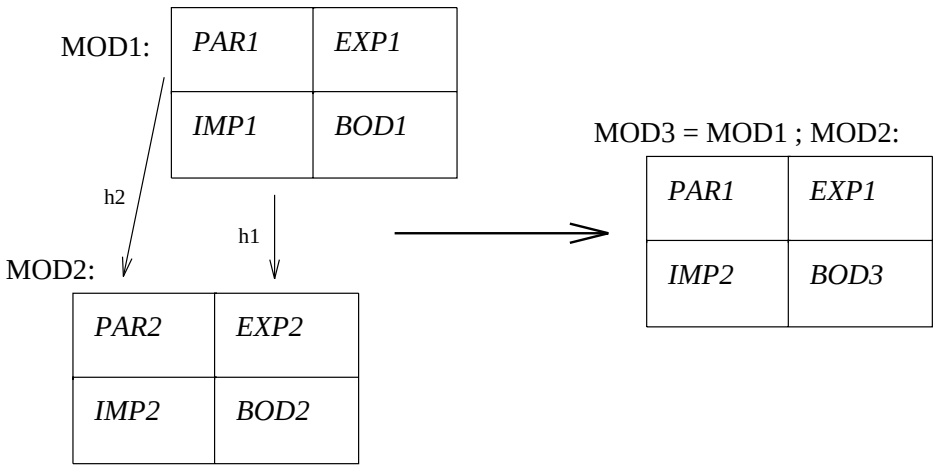


Fig. 2. Composition of modules

In this figure, the morphisms h_1, h_2 are the *composition morphisms* and BOD_3 denotes a pushout object of the inclusion morphism between IMP_1 and BOD_1 on the one hand, and the composition between h_1 and the inclusion morphism between EXP_2 and BOD_2 on the other hand.

As a main result for module specifications, we have shown in Ehrig and Mahr (1990) that the basic interconnection mechanisms preserve correctness and are compositional with respect to the semantics. The correctness of a modular system specification can thus be deduced from the correctness of its constituent parts, and the semantics can be composed from the semantics of its components, provided that the construction uses only the basic operations mentioned above.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Furthermore, there are nice compatibility results between these operations. This includes results for associativity, commutativity and distributivity (Ehrig and Mahr 1990). The relationship between algebraic module specifications and program modules is discussed in Löwe *et al.* (1991).

This theory of algebraic module specifications has been developed for the basic case of equational algebraic specifications based on total functions. In Ehrig and Mahr (1990) we have already indicated how to generalize it to abstract module specifications based on institutions in the sense of Goguen and Burstall (1984) and Goguen and Burstall (1992), which allows us to replace equational by other kinds of specifications. This extension of the theory is important in view of practical applications to the modular design of software systems.

In particular, it has turned out to be of practical significance to consider behaviour specifications in the sense of Reichel (1981), Sannella and Tarlecki (1987a) and Nivela and Orejas (1987), which are equational specifications with distinguished observational parts. In Orejas *et al.* (1989) we started to investigate semantic constructions for different categories of behaviour specifications in the sense of Nivela and Orejas (1987). We discovered that important semantic constructions like amalgamation and extension are not generally valid in the behavioural case. If, however, the underlying syntactic pushouts satisfy specific properties such as the *observation preserving property* (*cf.* Section 3), we can prove results regarding amalgamation and extension that are similar to the non-behavioural case. The remaining differences stem from the variety of possible specification morphisms. In the case of a specification morphism not necessarily preserving observable sorts (which nevertheless is necessary in some cases), we have no amalgamation but still a restricted form of extension. In fact, the main results for the composition of abstract module specifications only need extension.

In the remainder of this introduction, we give a historical and slightly more technical account of our results and an overview of the paper.

In 1991, when we prepared our invited paper Ehrig *et al.* (1991a) for AMAST'91, we recognized that behaviour specifications considered in the usual way do not satisfy the *satisfaction condition* required for institutions. Indeed, the theory of abstract module specifications does not require an explicit satisfaction condition at all. It is sufficient to start with a category $ASPEC$ of abstract specifications and a functor $Catmod$ assigning to each abstract specification $A\Sigma$ a model category $Catmod(A\Sigma)$. The objects A of $Catmod(A\Sigma)$ are considered to be models satisfying $A\Sigma$. From the categorical point of view, such a pair $(ASPEC, Catmod)$ with a contravariant functor $Catmod : ASPEC^{op} \to CAT$ into the super-category CAT of all categories is called an *indexed category* (Johnstone and Paré 1978; Tarlecki *et al.* 1991). In the context of a theory of abstract specifications, we prefer to call the pair $SF = (ASPEC, Catmod)$ a *specification frame*. Note that we used the alternative name *specification logic* in some of our previous papers (Ehrig 1989; Ehrig *et al.* 1989; Ehrig *et al.* 1991a; Ehrig *et al.* 1991b).

One main aim of the present paper is to present the theory of abstract module specifications. The corresponding material can be found in Section 4 and is based on specification frames introduced in Section 2. This theory requires only weak assumptions about existing free constructions, amalgamations and extensions. It generalizes the main

results of Ehrig and Mahr (1990) concerning correctness and compositionality of the basic interconnection mechanisms for module specifications from the equational specification frame to arbitrary specification frames satisfying these assumptions.

Another central aim of this paper is to present the theory of behaviour module specifications based on behaviour specifications in the sense of Nivela and Orejas (1987). For this purpose we introduce the corresponding behaviour specification frame *BEQSF* and study its properties concerning pushouts, free constructions, amalgamations and extensions in Section 3.

Because the behaviour specification frame satisfies the assumptions on the abstract level, we instantiate our theory of abstract module specifications to the behaviour case in Section 5. In particular, we 'prove' the correctness and compositionality of behaviour module specifications by mere instantiation of the corresponding theorems on the abstract level. These are important new results that have been only indicated in our AMAST'91 paper (Ehrig *et al.* 1991a). They show that our formulation of the theory of abstract module specifications is sensible.

In the conclusion we summarize our main ideas and discuss possible extensions.

2. A general specification framework

In this section we introduce a specification framework that generalizes the instances of, for example, the equational and the behavioural specification frameworks to be introduced in the following section. It is the basis for the definition of abstract, that is, institution independent, module specifications in Section 4 as a generalization of equational module specifications of Ehrig and Mahr (1990).

A well-known general specification framework is that of *institutions*. It was introduced in Goguen and Burstall (1984) and was further developed in Sannella and Tarlecki (1984), Meseguer (1989), and Goguen and Burstall (1992). The notion of institution captures abstract notions of *signatures*, *sentences* (*formulae*, *axioms*), *model functors*, *models* and *satisfaction*. It allows one to define a category of abstract *specifications* and a model functor that assigns to each abstract specification a category of models. In this way we obtain an indexed category in the sense of Tarlecki *et al.* (1991). Indexed categories are well known in category theory (Johnstone and Paré 1978) and are equivalent to fibered categories (Grothendieck 1963; Gray 1965; Benabou 1985).

Our theory of abstract module specifications does not require us to consider the satisfaction of sentences: it suffices to capture the assignment of model categories to specifications. Indexed categories are, therefore, the adequate framework for our purposes, and we choose to call them specification frames. We use this term because we employ indexed categories only to speak of abstract specifications $A\Sigma$, the corresponding model categories $Catmod(A\Sigma)$, and their morphisms and functors. In this scheme, the intuition is that every object A of $Catmod(A\Sigma)$ satisfies (the implicit axioms of) $A\Sigma$. But the notion of specification frames does indeed not make any satisfaction relation explicit.

In previous papers (Mahr 1989; Ehrig *et al.* 1989; Ehrig *et al.* 1991a; Ehrig *et al.* 1991b) we used the name *specification logic* rather than specification frame or indexed category. But we have been convinced by several colleagues that the name specification logic is

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

misleading. The reason is that notions like formulae, satisfaction and *deduction* are at most implicitly captured although they are available in all of the examples we have in mind. We hope that the name specification frame, which is also used in the context of abstract parameterized specifications (Ehrig and Große-Rhode 1994), avoids this confusion but captures the idea of having a general specification framework.

Of course we have to consider additional properties like pushouts, free constructions, amalgamations and extensions in order to formulate and prove structural properties of abstract specifications and their models in a specification frame.

In the following definition we restate the notion of an indexed category in the context of abstract specifications, and introduce some terminology. The relation of specification frames to institutions in the sense of Goguen and Burstall (1984) and Sannella and Tarlecki (1984) is made explicit in Fact 2.2.

Definition 2.1. (Indexed category/specification frame) An *indexed category/specification frame* is a pair $(ASPEC, Catmod)$, where $ASPEC$ is a *category of abstract specifications* and $Catmod$ is a contravariant *model functor* from $ASPEC$ into the super-category CAT of all categories. Given an *abstract specification* $A\Sigma$, that is, an object of $ASPEC$, a $A\Sigma$-*model* is an object of $Catmod(A\Sigma)$; given a *specification morphism* f, that is, a morphism in $ASPEC$, the $Catmod$-image of f is called the *forgetful functor* (with respect to f) and is denoted by V_f.

The loss of explicit reference to sentences and satisfaction makes it necessary to require specification frames to have additional syntactic and semantic properties. We also applied this idea in Ehrig *et al.* (1991a) and Ehrig *et al.* (1991b), where we considered *generalized morphisms* and new concepts for *generalized amalgamation* and *extension*. However, before we introduce the properties that are needed for our present purposes, we will give some examples of concrete specification frames.

1 The *equational specification frame* $EQSF = (SPEC, EQCatmod)$ consists of the category $SPEC$ of equational algebraic specifications and the functor $EQCatmod$ that assigns to each specification SP the category $Alg(SP)$ of Algebras of SP, that is, $EQCatmod(SP) =_{def} Alg(SP)$. If we replace equations by conditional equations, we obtain the *conditional equational specification frame* $CEQSF$.

2 The *behaviour equational specification frame* $BEQSF = (BSPEC, Beh)$ consists of the category $BSPEC$ of equational behaviour specifications and the functor Beh that assigns to each behaviour specification $B\Sigma$ the category $Beh(B\Sigma)$ of algebras satisfying $B\Sigma$ behaviourally (Nivela and Orejas 1987; Orejas *et al.* 1989). There is no institution that generates $BEQSF$ in the way described in Fact 2.2. The behaviour equational specification frame thus constitutes an important example of a specification formalism that can be studied as a specification frame but not as an institution. The details of this observation can be found in Section 5.

3 A related example of another specification frame that cannot be seen as an institution in the obvious way is $VIEWSF = (View\text{-}BSPEC, Beh)$ (Cornelius 1990a; Orejas *et al.* 1989). The main difference between $BSPEC$ and $View\text{-}BSPEC$ is the choice of the notion of specification morphisms, namely the degree of preservation of the observable

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

and non-observable sorts, respectively. We do not give *VIEWSF* in detail in this paper but discuss its relevant properties in Section 5, Remark 5.1.

4 The *projection specification frame* $PROSF = (PROSPEC, PCatmod)$ consists of the category *PROSPEC* of projection specifications and of the functor *PCatmod*. It assigns to each projection specification $P\Sigma$ the category $Cat_{compl,sep}(P\Sigma)$ of all complete and separate projection algebras satisfying $P\Sigma$ (Ehrig *et al.* 1990; Große-Rhode 1989).

5 In addition to these examples there are several others based on different kinds of axioms, such as universal Horn or full first-order formulae, order-sorted signatures and constraints on the syntactic level. On the semantic level, different kinds of algebras and structures are possible, such as partial or continuous algebras, or models of first-order logic.

The *first-order specification frame FOSF*, for example, has first-order signatures and axioms on the syntactic level, and the corresponding models on the semantic level.

In the following fact we state the *standard specification frame* that corresponds to a given institution.

Fact 2.2. (Specification frames induced by institutions) Let $I = (SIGN, Sen, Mod, \models)$ be an institution. Then the *induced specification frame* $SF(I)$ is given by $SF(I) = (ASPEC, Catmod)$.

The objects of *ASPEC* are the I-presentations (Σ, S), $\Sigma \in SIGN$ and $S \subseteq Sen(\Sigma)$. The morphisms of *ASPEC* are the usual presentation morphisms of, for example, Goguen and Burstall (1992). $Catmod(\Sigma, S)$ is the full subcategory of $Cat(\Sigma)$ comprising the models satisfying (via $\models$) the set of sentences S.

As already indicated, specification frames need to have several syntactic and semantic properties for further investigation. The first kind of property is related to left adjoints of the forgetful functors of specification morphisms.

Definition 2.3. (Free constructions, strong persistency, liberality) Let $SF = (ASPEC, Catmod)$ be a specification frame:

1 For a specification morphism $f : A\Sigma_1 \to A\Sigma_2 \in ASPEC$, we say a functor $F : Catmod(A\Sigma_1) \to Catmod(A\Sigma_2)$ is *strongly persistent* if $V_f \circ F$ is the identity.

2 A specification morphism $f \in ASPEC$ is said to be *(strongly) liberal* if the forgetful functor V_f has a (strongly persistent) left adjoint F_f.

3 *SF* is said to have *free constructions* if each morphism $f \in ASPEC$ is liberal.

From a categorical point of view, it might appear more natural to define strong persistency of a left adjoint as the property of the unit η of the adjunction $F_f \dashv V_f$ to be the identity transformation. In fact, this alternative is equivalent to the definition above (see the Appendix). Moreover, there is just one place where we could use it (in the proof of Theorem 2.12), but then we invoke the equivalence result.

Remark 2.4. (Composition of strongly persistent functors) It is a well-known fact that the classes of strongly persistent and left adjoint functors are respectively closed under composition.

The second kind of property of specification frames is based on a designated class of pushout diagrams in the category *ASPEC* of a specification frame. This technique enables

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

us to extend the properties of *amalgamation*, that is, the continuity of the model functor with respect to pushouts, and *extension* from standard specification frames such as *EQSF* to less standard ones such as *BEQSF* and *VIEWSF*. In these cases, the problem with a conventional theory of abstract module specifications without a designated pushout class is that not all pushouts of behaviour or view specification morphisms have amalgamations and/or extensions. Certain practically interesting sub-classes of these classes of pushouts, however, do possess amalgamations and/or extensions. The idea is to reflect this situation at the abstract level and, thereby, achieve the desired generality. As an artifact, it becomes possible to instantiate the theory to specification frames with empty classes of designated pushouts. Such applications are of course trivial and do not yield anything useful. On the other hand, this aspect does not affect in any way the possibility of carrying out instantiations to specification frames with non-trivial and practically interesting classes of designated pushouts. So the possibility of trivial instantiations does not present any problem.

Definition 2.5. (Specification frames with pushouts) A *specification frame with pushouts* (SF, PO) consists of a specification frame $SF = (ASPEC, Catmod)$ and a class of pushout diagrams PO in $ASPEC$.

Definition 2.6. (The maximum pushout class *PO_ALL*) $PO\_ALL(ASPEC)$ denotes the class of all pushout diagrams in any given category $ASPEC$ of abstract specifications.

Note that we do not *require* the syntactic category of a specification frame to be cocomplete or to contain any restricted class of colimits. The syntactic categories of the specification frames that we are interested in, however, *are* all cocomplete.

Definition 2.7. (Amalgamation) A specification frame with pushouts (SF, PO) is said to *have amalgamations* if $Catmod$ preserves every pushout in PO.

Because of the contravariance of $Catmod$, an *ASPEC-pushout diagram* as in Figure 3 is translated to a *pullback diagram* in CAT (Figure 4). This property is expanded in Fact 2.8 below.

$$\begin{array}{ccc} A\Sigma_0 & \xrightarrow{f_1} & A\Sigma_1 \\ {\scriptstyle f_2}\downarrow & & \downarrow{\scriptstyle g_1} \\ A\Sigma_2 & \xrightarrow[g_2]{} & A\Sigma_3 \end{array}$$

Fig. 3. *ASPEC*-pushout diagram

$$\begin{array}{ccc} Catmod(A\Sigma_0) & \xleftarrow{V_{f_1}} & Catmod(A\Sigma_1) \\ {\scriptstyle V_{f_2}}\uparrow & & \uparrow{\scriptstyle V_{g_1}} \\ Catmod(A\Sigma_2) & \xleftarrow[V_{g_2}]{} & Catmod(A\Sigma_3) \end{array}$$

Fig. 4. *CAT*-pullback diagram

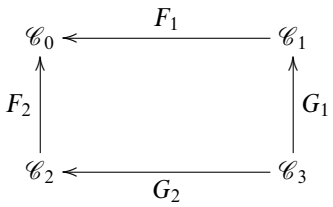
The name amalgamation has its origin in the situation that prevails in standard algebraic specification frames such as *EQSF*. In these settings it makes sense to regard an algebra

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

A_3 in $Catmod(A\Sigma_3)$ as the *amalgamated sum* of its forgetful functor images $V_{g_1}(\mathsf{A}_3)$ and $V_{g_2}(\mathsf{A}_3)$. See Ehrig and Mahr (1985) for the converse construction on the basis of algebras A_1 and A_2 belonging to $Catmod(A\Sigma_1)$ or $Catmod(A\Sigma_2)$, respectively.

The importance of amalgamation for the definition of the semantics of combined specifications is due to the following well-known characterization of pullback diagrams in *CAT*. For a proof of its forward direction see, for example, Ehrig and Mahr (1985).

Fact 2.8. (Characterization of pullbacks in *CAT*) A commutative diagram of the form



in *CAT* is a pullback if and only if the following properties hold:

1 For all objects A_1 and A_2 in $\mathscr{C}_1$ or $\mathscr{C}_2$, respectively: whenever $F_1(\mathsf{A}_1) = F_2(\mathsf{A}_2)$, there exists a unique object A_3 in $\mathscr{C}_3$ such that $G_i(\mathsf{A}_3) = \mathsf{A}_i$, $i = 1, 2$.
2 For all morphisms h_1 and h_2 in $\mathscr{C}_1$ or $\mathscr{C}_2$, respectively: whenever $F_1(h_1) = F_2(h_2)$, there exists a unique morphism h_3 in $\mathscr{C}_3$ such that $G_i(h_3) = h_i$, $i = 1, 2$.

We will apply Fact 2.8 only to situations such as in Figures 3 and 4, where a specification frame with pushouts (SF, PO) is given and the pullback in *CAT* is the *Catmod*-image of a pushout in *PO*. The objects of (1) and the morphisms of (2) are, therefore, always thought of as models or morphisms between models. Furthermore, because of the uniqueness of A_3 and h_3, it is possible to introduce the following functional notation:

$$\mathsf{A}_1 +_{(f_1,f_2)} \mathsf{A}_2 =_{def} \mathsf{A}_3 \text{ and } h_1 +_{(f_1,f_2)} h_2 =_{def} h_3.$$

For models, the notation is defined if A_i, $i = 1, 2$, is a $Catmod(A\Sigma_i)$-model and $V_{f_1}(\mathsf{A}_1) = V_{f_2}(\mathsf{A}_2)$. Then it designates the unique $A\Sigma_3$-model A_3 with $V_{g_i}(\mathsf{A}_3) = \mathsf{A}_i$, which exists due to Fact 2.8(1). Everything is analogous for homomorphisms.

The following properties of the amalgamation operator are immediate:

Fact 2.9.

1 For every $A\Sigma_3$-model A_3 and every morphism h_3 between such models,

$$\mathsf{A}_3 = V_{g_1}(\mathsf{A}_3) +_{(f_1,f_2)} V_{g_2}(\mathsf{A}_3) \text{ and } h_3 = V_{g_1}(h_3) +_{(f_1,f_2)} V_{g_2}(h_3).$$

2 Given that h_i is a morphism from A_i to A'_i, $i = 1, 2$, then $h_1 +_{(f_1,f_2)} h_2$ is a morphism from $\mathsf{A}_1 +_{(f_1,f_2)} \mathsf{A}_2$ to $\mathsf{A}'_1 +_{(f_1,f_2)} \mathsf{A}'_2$.

In the following lemma, we extend the concept of amalgamation from models and morphisms to functors.

Lemma 2.10. (Amalgamation of functors) Let (SF, PO) be a specification frame with pushouts and amalgamations, and let two particular pushouts in PO and *ASPEC*-morphisms $\varphi_i : A\Sigma_i^1 \to A\Sigma_i^2$, $i \in \{0, 1, 2, 3\}$ be given as shown in Figure 5 such that the whole diagram commutes. Moreover, let $F_i : Catmod(A\Sigma_i^1) \to Catmod(A\Sigma_i^2)$, $i = 1, 2$, be functors such that (see Figure 6)

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

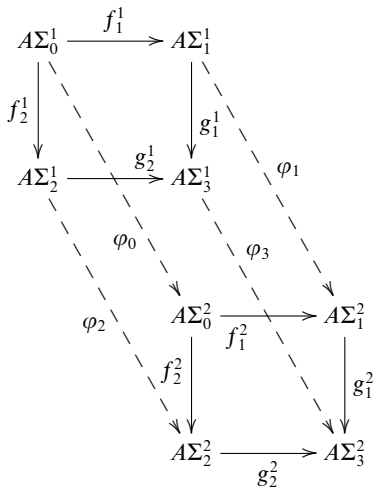


Fig. 5. The underlying syntax diagram for amalgamated functors

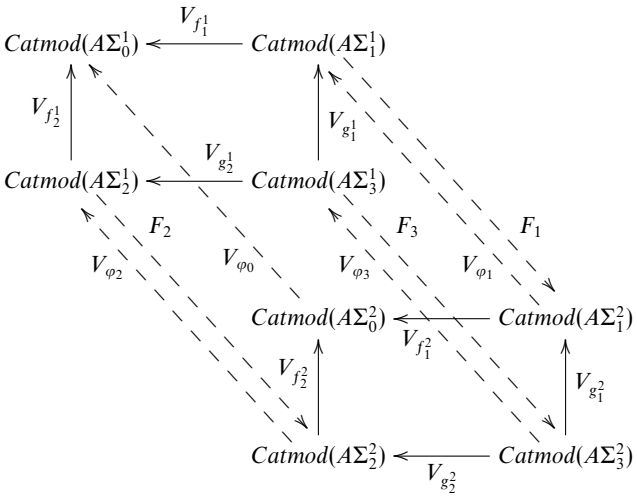


Fig. 6. The underlying semantics diagram for amalgamated functors

$$V_{f_2^2} \circ F_2 \circ V_{g_2^1} = V_{f_1^2} \circ F_1 \circ V_{g_1^1}.$$

Then we can define the *amalgamated sum of* F_1 *and* F_2, written F_3 or $F_1 +_{(f_1^2, f_2^2)} F_2$, for every $\mathsf{A}_3 \in Catmod(A\Sigma_3^1)$ by

$$F_3(\mathsf{A}_3) =_{def} (F_1 \circ V_{g_1^1}(\mathsf{A}_3)) +_{(f_1^2, f_2^2)} (F_2 \circ V_{g_2^1}(\mathsf{A}_3)),$$

and similarly for morphisms. Then we have:

1 F_3 is the unique functor satisfying $V_{g_i^2} \circ F_3 = F_i \circ V_{g_i^1}$, $i = 1, 2$.
2 If $F_i \dashv V_{\varphi_i}$, $i = 1, 2$, then $F_3 \dashv V_{\varphi_3}$.
3 If F_i, $i = 1, 2$, are strongly persistent, then so is F_3.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Proof.

1 This part is a direct consequence of the amalgamation properties, Fact 2.8.

2 We will restrict ourselves to constructing the unit η_3 of the adjunction $F_3 \dashv V_{\varphi_3}$, using the units η_1 and η_2 of the adjunctions given in the premise. The rest of the proof is straightforward:

$$\eta_3(\mathsf{A}_3) =_{def} \eta_1(V_{g_1^1}(\mathsf{A}_3)) +_{(f_1^1, f_2^1)} \eta_2(V_{g_2^1}(\mathsf{A}_3)),$$

where the domain and co-domain objects are as follows: $\eta_i(V_{g_1^i}(\mathsf{A}_3)) : V_{g_1^i}(\mathsf{A}_3) \to V_{\phi_i} \circ F_i \circ V_{g_1^i}(\mathsf{A}_3)$, $i = 1, 2$, and $\eta_3(\mathsf{A}_3) : \mathsf{A}_3 \to V_{\phi_3} \circ F_3(\mathsf{A}_3)$.

3 By the following rewriting sequence:

$$\begin{aligned}
V_{\varphi_3} \circ F_3(\mathsf{A}_3) &= (V_{g_1^1} \circ V_{\varphi_3} \circ F_3(\mathsf{A}_3)) +_{(f_1^1, f_2^1)} (V_{g_2^1} \circ V_{\varphi_3} \circ F_3(\mathsf{A}_3)) \\
&\qquad \text{by Fact 2.9} \\
&= (V_{\varphi_1} \circ V_{g_1^2} \circ F_3(\mathsf{A}_3)) +_{(f_1^1, f_2^1)} (V_{\varphi_2} \circ V_{g_2^2} \circ F_3(\mathsf{A}_3)) \\
&\qquad \text{since } V_{g_i^1} \circ V_{\varphi_3} = V_{\varphi_i} \circ V_{g_i^2},\ i = 1, 2 \\
&= (V_{\varphi_1} \circ F_1 \circ V_{g_1^1}(\mathsf{A}_3)) +_{(f_1^1, f_2^1)} (V_{\varphi_2} \circ F_2 \circ V_{g_2^1}(\mathsf{A}_3)) \\
&\qquad \text{by (1)} \\
&= V_{g_1^1}(\mathsf{A}_3) +_{(f_1^1, f_2^1)} V_{g_2^1}(\mathsf{A}_3) \\
&\qquad \text{by strong persistency} \\
&= \mathsf{A}_3 \qquad \text{by Fact 2.9}
\end{aligned}$$

□

The following definition of free extension again refers to the class *PO*. It explains the property that is crucial for the compositionality and correctness results of the module operations to be introduced in Section 4.

Definition 2.11. (Free extensions) A specification frame with pushouts (SF, PO) is said to *have free extensions* if for every pushout diagram in *PO*, as illustrated in Figure 3, the following property holds: if F with $F \dashv V_{f_1}$ is strongly persistent, then g_2 is strongly liberal, that is, there exists a strongly persistent functor $Ext_{f_2}(F)$ with $Ext_{f_2}(F) \dashv V_{g_2}$ such that the diagram in Figure 7 commutes.

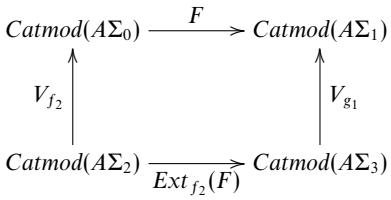


Fig. 7. Free extension of F

It is well known that *EQSF* and *CEQSF* have amalgamations and free extensions for the maximum classes *PO_ALL(SPEC)* and *PO_ALL(SPEC)* of all pushout diagrams in the underlying syntactic categories *SPEC* and *CSPEC* (see Remark 2.6). The behaviour specification frame *BEQSF* of the following section is an example of a specification frame with a restricted class of syntactic pushout diagrams providing amalgamations and extensions.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Theorem 2.12. (Extension by amalgamation) If a specification frame with pushouts (SF, PO) has amalgamations, then it has free extensions.

Proof. Let (SF, PO) be a specification frame with pushouts and amalgamations. We need to show that (SF, PO) has free extensions. So, given a pushout diagram in PO as depicted in Figure 3, let F in Figure 8 be strongly persistent.

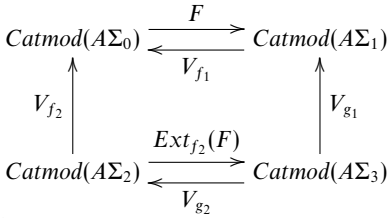


Fig. 8. Complete *Catmod*-diagram

First of all, we define $Ext_{f_2}(F)$ building amalgamated sums according to Fact 2.8:

$$Ext_{f_2}(F)(\mathsf{A}_2) =_{def} (F \circ V_{f_2}(\mathsf{A}_2)) +_{(f_1,f_2)} \mathsf{A}_2$$
$$Ext_{f_2}(F)(h_2) =_{def} (F \circ V_{f_2}(h_2)) +_{(f_1,f_2)} h_2$$

for all A_2 and h_2 in $Catmod(A\Sigma_2)$. This is well defined because F is strongly persistent, that is, $V_{f_1} \circ F \circ V_{f_2}(\mathsf{A}_2) = V_{f_2}(\mathsf{A}_2)$, and similarly for h_2.

Now we have to show that $Ext_{f_2}(F)$ is strongly persistent and left adjoint to V_{g_2}. The persistency property, $V_{g_2} \circ Ext_{f_2}(F) = Id_{Catmod(A\Sigma_2)}$, is a direct consequence of the definition of $Ext_{f_2}(F)$ in combination with Fact 2.8.

It remains to show the adjunction $Ext_{f_2}(F) \dashv V_{g_2}$. We do so by using id_{A_2} as the unit for every A_2 in $Catmod(A\Sigma_2)$, and then constructing a unique h^* such that the triangle in Figure 9 commutes for any given h. In other words, we construct a unique

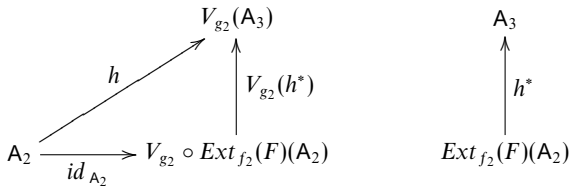


Fig. 9. The adjunction $Ext_{f_2}(F) \dashv V_{g_2}$

$h^* : Ext_{f_2}(F)(\mathsf{A}_2) \to \mathsf{A}_3$ such that $V_{g_2}(h^*) = h$.

The first step consists in exploiting the adjunction $F \dashv V_{f_1}$, as shown in Figure 10, so as to obtain a unique h_2^* such that $V_{f_1}(h_2^*) = V_{f_2}(h)$. This use of $F \dashv V_{f_1}$ is possible

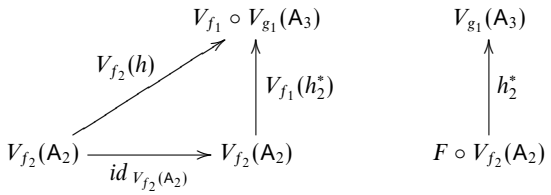


Fig. 10. Using the adjunction $F \dashv V_{f_1}$

because $g_1 \circ f_1 = g_2 \circ f_2$ implies $V_{f_1} \circ V_{g_1}(\mathsf{A}_3) = V_{f_2} \circ V_{g_2}(\mathsf{A}_3)$, the persistency of F implies

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

$V_{f_2}(\mathsf{A}_2) = V_{f_2} \circ F \circ V_{f_2}(\mathsf{A}_2)$ and the property proved in the appendix allows us to assume $\eta(V_{f_2}(\mathsf{A}_2)) = id_{V_{f_2}(\mathsf{A}_2)}$.

The second step consists of stating $h^* =_{def} h_2^* +_{(f_1,f_2)} h$, which is possible by the very property $V_{f_1}(h_2^*) = V_{f_2}(h)$.

The third step consists of showing $V_{g_2}(h^*) = h$, but this property is actually an immediate consequence of the definition of h^*.

The last step consists of showing that h^* is unique with respect to $V_{g_2}(h^*) = h$, so assume $h' : Ext_{f_2}(F)(\mathsf{A}_2) \to \mathsf{A}_3$ is a morphism with $V_{g_2}(h') = h$. This property implies $V_{f_1} \circ V_{g_1}(h') = V_{f_2}(h)$ so, by the uniqueness of h_2^* with respect to $V_{f_1}(h_2^*) = V_{f_2}(h)$, we have $V_{g_1}(h') = h_2^*$. At the same time, h^* is defined as the amalgamated sum $h_2^* +_{(f_1,f_2)} h$, so it is unique with respect to $V_{g_1}(h^*) = h_2^*$ and $V_{g_2}(h^*) = h$, so $h' = h^*$. □

3. The behaviour specification frame

Behaviour semantics of algebraic specifications was introduced with the aim of following more closely the methodological principles underlying information hiding in software engineering. The syntactical idea is to denote some of the sorts of a specification as *observable* and the others as *non-observable*. The corresponding semantical idea is to admit as algebraic models of the specification those that satisfy the (generally infinite) class of *observable consequences* of the equations. Intentionally, this means that every *correct implementation* of the non-observable part is a model of the specification. Another semantical consequence of this approach is that two models are said to be *equivalent* if and only if they provide the same *observable behaviour*.

A typical example will motivate the approach. Consider the following specification for sets of elements:

$SET(ELEM)$: **sorts:** $Bool, Nat, Elem$ (observable)
Set (non-observable)
opns: [...] *(the Booleans)*
[...] *(the natural numbers)*
$eq : Elem, Elem \to Bool;$
$\varnothing : \to Set;$
$in : Elem, Set \to Set;$
$is\_in : Elem, Set \to Bool;$
$card : Set \to Nat;$
vars: $e, e_1, e_2 : Elem; s : Set$
eqns: $in(e_1, in(e_2, s)) = in(e_2, in(e_1, s));$
$in(e, in(e, s)) = in(e, s);$

$is\_in(e, \varnothing) = false;$
$is\_in(e, in(e, s)) = true;$
$eq(e_1, e_2) = false \ \to \ is\_in(e_1, in(e_2, s)) = is\_in(e_1, s);$

$card(\varnothing) = 0;$
$is\_in(e, s) = false \ \to \ card(in(e, s)) = succ(card(s));$
$is\_in(e, s) = true \ \to \ card(in(e, s)) = card(s);$

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Sets are containers that *behave* in a particular way: the insertion order of the elements does not matter, and each element exists at most once in a set.

Now, even in mathematical notation we have no unique *representation* of sets. For example, $\{1, a\}$ and $\{a, 1\}$ represent the same set. No one really cares about the representation as long as the intended (observational) behaviour is as expected. This behaviour can be expressed by the class of *computations* over set-terms that yield values of observational sort. We designate the sorts $Bool, Nat, Elem$ as observable because their elements are supposed to be 'representation sensitive' in the sense that the syntax is sufficient to decide equality of elements.

For the specification $SET(ELEM)$ some behavioural consequences of the second equation (built over additional variables of observable sorts) are given in the following remark.

Remark 3.1. (Examples of behavioural consequences)

$$\begin{aligned} is\_in(x, \mathbf{in}(\mathbf{e}, \mathbf{in}(\mathbf{e}, \varnothing))) &= is\_in(x, \mathbf{in}(\mathbf{e}, \varnothing)) \\ is\_in(x, in(y, \mathbf{in}(\mathbf{e}, \mathbf{in}(\mathbf{e}, \varnothing)))) &= is\_in(x, in(y, \mathbf{in}(\mathbf{e}, \varnothing))) \\ card(\mathbf{in}(\mathbf{e}, \mathbf{in}(\mathbf{e}, \varnothing))) &= card(\mathbf{in}(\mathbf{e}, \varnothing)) \\ card(in(x, \mathbf{in}(\mathbf{e}, \mathbf{in}(\mathbf{e}, \varnothing)))) &= card(in(x, \mathbf{in}(\mathbf{e}, \varnothing))) \end{aligned}$$

These examples illustrate the underlying principle of building observable computations over a given non-observable term. Observability of a term has two facets:

— *Output observability*: this means that the term is of an observable sort, that is, the evaluation of the term in an algebra *outputs* an observable value;

— *Input observability*: this means that the variables the term is constructed over are all of observable sorts, that is, the evaluation of the term in an algebra only requires observable *input*.

An observable computation is both input and output observable. That is: in order to build an observable computation over a non-observable term, we first have to embed it into a *context* (term) of observable sort. This context term describes how observable information about a non-observable term is accessed. Secondly, input observability is achieved by substituting each non-observable variable by an input-observable term (*cf.* Definition 3.4).

Now consider the following two algebras A, B of the signature of $SET(ELEM)$:

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

SET(*ELEM*)	A	B
Bool	$\{true, false\}$	$\{true, false\}$
Nat	$\mathbb{N}$	$\mathbb{N}$
Elem	$\{a, \ldots, z\}$	$\{a, \ldots, z\}$
Set	$\mathscr{P}(\{a, \ldots, z\})$	$\{a, \ldots, z\}^*$
(*Bool*-ops)	*(standard)*	*(standard)*
(*Nat*-ops)	*(standard)*	*(standard)*
eq	*Equality*	*Equality*
$\varnothing$	$\varnothing$	λ
in	$in_{\mathsf{A}}(\alpha, s) =_{def} \{\alpha\} \cup s$	$in_{\mathsf{B}}(\alpha, l) =_{def} \alpha.l$
is_in	$\in$	*string membership*
card	*set cardinality*	*number of distinct elements*

In the classical algebraic sense, these two algebras are not isomorphic. B does not even satisfy the equations of *SET(ELEM)*. But the given algebra of sequences B is a typical *correct implementation* of the specification. It is therefore a behavioural model of *SET*(*ELEM*), it satisfies, for example, each of the behavioural consequences in Remark 3.1 above. This means, the 'implementation' of the *Set*-carrier, that is, the list representation, *behaves* exactly the way we expect for sets, if tested in terms of the obsevable sorts *Bool, Nat* and *Elem*. Moreover, A and B are behaviourally equivalent in the sense to be explained in this section.

Early work on algebraic specifications (Giarratana *et al.* 1976; Guttag and Horning 1978) already recognized the need for such a kind of semantics. At that time the proposed solution was final semantics (Wand 1979). The first true approaches to behaviour semantics are due to Reichel (Reichel 1981; Reichel 1984b) and Goguen and Meseguer (Goguen and Meseguer 1982; Meseguer and Goguen 1985). Further work was done later by Bidoit, Hennicker, and Wirsing (Hennicker and Wirsing 1985; Bidoit *et al.* 1994), Sannella and Tarlecki (Sannella and Tarlecki 1987b) and Nivela and Orejas (Nivela 1987; Nivela and Orejas 1987; Orejas *et al.* 1989).

In this section we develop the theory of behaviour specifications based upon the results of Orejas *et al.* (1989). The notions presented lead to the definition of the behaviour equational specification frame *BEQSF* in Definition 3.6. Then we show the general existence of syntactical pushouts in *BEQSF* and prove that *BEQSF* has free constructions. Finally, we show how the class of syntactical pushouts can be restricted to a class PO_{Beh} in order to make $(BEQSF, PO_{Beh})$ have amalgamations and extensions.

We first need to define the ingredients of the specification frame.

Definition 3.2. (Behaviour specification)

1 A *behavioural* specification $B\Sigma = (Obs, \Sigma)$ consists of an equational specification $\Sigma = (S, OP, E)$ together with a designated subset $Obs \subseteq S$. The elements of *Obs* are called *observable sorts* of $B\Sigma$.
2 Given an equational specification morphism $\varphi : \Sigma_1 \to \Sigma_2$, that is, satisfying $E_2 \models \varphi(E_1)$ in the standard institution of many–sorted equational logic. A *behaviour (equa-*

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

tional) specification morphism $\varphi : (Obs_1, \Sigma_1) \to (Obs_2, \Sigma_2)$ additionally preserves, in its sort mapping component, both the observable and the non-observable sorts, that is: $\varphi(Obs_1) \subseteq Obs_2$ and $\varphi(S_1 - Obs_1) \subseteq S_2 - Obs_2$.

3 *BSPEC* denotes the category of behaviour specifications. Its objects and morphisms are defined above in this definition.

In the following we will abbreviate the set of non-observable sorts of a behaviour specification by $NonObs =_{def} S - Obs$.

Remark 3.3. (Different notions of behaviour morphisms) The restricted requirement for a behaviour specification to preserve both parts *Obs* and *NonObs* of the domain specification is not the only reasonable choice. In Orejas *et al.* (1989) the two sensible alternatives that each only requires one of the two properties in Definition 3.2(2) to be satisfied, are discussed in detail. The more important class of morphisms requires the preservation of the non-observable sorts only. A member of this class is called a *view-morphism*. Its importance is of a methodological nature regarding iterated parameter passing. In the example *SET*(*ELEM*) the sort *Elem* is naturally chosen to be observable. But if we want to actualize the set by itself in order to construct sets of sets of things we need to map the observable sort *Elem* to the non-observable sort *Set*. This implies that such an actualization morphism does not preserve the observational part and should thus be considered a view morphism.

Every algebra A of a behaviour signature $B\Sigma = (Obs, \Sigma)$ (a behaviour specification without equations) corresponds naturally to an algebra of the standard algebraic signature Σ. Behavioural satisfaction of equations by A is defined as follows in terms of the standard equational satisfaction of equations by A seen as a Σ-algebra.

Definition 3.4. (Observable context, behaviour satisfaction) Given a behaviour specification $B\Sigma = (Obs, \Sigma)$ with $\Sigma = (S, OP, E)$. Let $X_{Obs} = (X_s)_{s \in Obs}$ denote a family of sets of variables of observable sort. In addition, let $s \in S$ be a sort in $B\Sigma$.

1 An *observable* $B\Sigma$*-context over* s is a $B\Sigma$-term $c[z] \in T_\Sigma(X_{Obs} + \{z\})$ of observable sort that is built upon the observable variables in X_{Obs} and the additional variable $z \notin X_{Obs}$ of sort s.

2 Given a term $t \in T_\Sigma(X)$ over an arbitrary set X of variables, we define *the application of an observable context* $c[z]$ *to* t by the following substitution:

$$c[t] =_{def} c[z]\{t/z\}.$$

3 A Σ-algebra A is said to *behaviourally satisfy* a Σ-equation $e \equiv (X; l = r)$ of sort s iff for each observable $B\Sigma$-context $c[z]$ over s and for every assignment $\sigma : X \to T_\Sigma(X_{Obs})$, the algebra A satisfies (in the standard equational sense) the equation

$$(X_{Obs}; c[\sigma^*(l)] = c[\sigma^*(r)]).$$

(Here, $\sigma^* : T_\Sigma(X) \to T_\Sigma(X_{Obs})$ denotes the free extension of the variable assignment σ to arbitrary terms.) We then write $\mathsf{A} \models_{Beh} e$.

This definition provides us with the tools to build formally the observable computations over arbitrary terms mentioned in the discussion after Remark 3.1: the application of an observable context yields output observability while the application of σ^* yields input

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

observability. Thus, the equations required to be satisfied according to the third point of the definition are the results of equating the possible observable computations over the constituting terms l and r.

Behavioural satisfaction relies on the standard equational satisfaction $\models$, which is the satisfaction relation in the equational algebraic institution. Behavioural satisfaction can be generalized to be dependent of an arbitrary algebraic institution (Bidoit *et al.* 1994).

Now, we can see what it means to say that the algebra B of the introducing example satisfies behaviourally the first two equations of $SET(ELEM)$. In the list of four *behavioural consequences* in Remark 3.1 we first had to substitute the non-observable variable s in order to eliminate every non-observable variable. We did this by substituting '$\varnothing$' in every case. The context terms used in the four examples were:

$$is\_in(x, z)$$
$$is\_in(x, in(y, z))$$
$$card(z)$$
$$card(in(x, z)).$$

Of course, this is only a small subset of the infinite set of equations that B has to satisfy, but it is enough to illustrate the principle.

Reichel in Reichel (1981) defines a notion of behavioural satisfaction that differs with respect to input observability. That is, he defines (here using the notations of Definition 3.4) $\mathsf{A} \models_{Beh} e$ iff for all observable $B\Sigma$-contexts $c[z]$ of the sort of e we have $\mathsf{A} \models c[e]$.

The following example illustrates the difference between the two notions. Consider the $SET(ELEM)$-algebra C that we get if we extend the algebra A in its Set-carrier:

$$\mathsf{C}_{Set} =_{def} \mathsf{A}_{Set} \cup \{*\}.$$

Here, we call this new element '$*$' a *junk* element because we cannot name it syntactically: there is no ground (that is, variable free) term evaluating to $*$. That is, we have not explicitly *specified* it.

The operations $in_{\mathsf{C}}, is\_in_{\mathsf{C}}$ and $card_{\mathsf{C}}$ are extended as follows:

$$in_{\mathsf{C}}(\_, *) =_{def} *$$
$$is\_in_{\mathsf{C}}(\_, *) =_{def} false$$
$$card_{\mathsf{C}}(*) =_{def} 0.$$

Now, in the sense of Reichel, C does not (behaviourally) satisfy the following equation:

$$is\_in(x, in(x, s)) = true$$

Because of the operations that yield non-observable sorts, we can naturally *generate* non-observable elements, but the junk-elements of non-observable sorts cannot be addressed directly. This is the reason for calling it non-observable. Therefore C should indeed be considered a behavioural model of the above equation. Our definition guarantees this.

For the definition of *BEQSF* we still have to define the behavioural notion of homomorphisms. For this we have to take into account the fact that the non-observable *structure* does not have to be preserved directly.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Definition 3.5. (Behaviour model, $B\Sigma$-behaviour morphism) Given a behaviour specification $B\Sigma = (Obs, \Sigma)$ and a Σ-algebra A. Let X_{Obs} be a family of sets of variables of observable sorts in Σ.

1 A is called a $B\Sigma$-*algebra* iff it behaviourally satisfies each of the equations in Σ.

2 A term $t \in T_\Sigma(X_{Obs})$ of observable sort is called an *observable* $B\Sigma$-*computation* if it is not a variable. It is called *minimal* if none of its sub-terms is an observable $B\Sigma$-computation.

3 Given an observable $B\Sigma$-computation t, the set of *observable computations included in* t, denoted by $Min(t)$, is defined recursively by:

(a) if t is minimal, then $Min(t) =_{def} \{t\}$;

(b) otherwise, at some position u, t has a sub-term t_0 that is a minimal observable $B\Sigma$-computation. Let x be a new variable of the sort of this sub-term. Then we define:

$$Min(t) =_{def} \{t_0\} \cup Min(t[u \leftarrow x])$$

(by $t[u \leftarrow x]$ we mean the replacement of x for t_0, that is, the term we find in t at its u-position).

The definition of $Min(t)$ is well defined as long as the assignment of the term positions in t to the new variables (here denoted by 'x') that are introduced during the recursive procedure, is a pre-given *functional* relation (and an *injective* one, otherwise, the requirement for the variable to be 'new', that is, unused in the remaining context, cannot be satisfied).

4 Let Obs be the set of observable sorts and let $\mathsf{A}_{Obs} = (\mathsf{A}_s \mid s \in Obs)$ denote the family of observable carriers in A. Let $T_\Sigma(\mathsf{A}_{Obs})$ be the Σ-algebra freely generated by A_{Obs}. (Note, technically, this is only defined for S-indexed families – thus one really wants A_{Obs} to be the S-indexed family with $\mathsf{A}_s = \varnothing$ when s is a non-observable sort.)
A term $t \in T_\Sigma(\mathsf{A}_{Obs})$ of observable sort is called an *observable* $B\Sigma$-*computation over* A.

5 Given two $B\Sigma$-algebras, A and B, define a $B\Sigma$-*behaviour morphism* $f : \mathsf{A} \to \mathsf{B}$ to be an Obs-indexed family of mappings, $f = (f_s : \mathsf{A}_s \to \mathsf{B}_s)_{s \in Obs}$ such that, where $f^\# : T_\Sigma(\mathsf{A}_{Obs}) \to T_\Sigma(\mathsf{B}_{Obs})$ is the unique Σ-homomorphism extending f, and $\epsilon_\mathsf{A} : T_\Sigma(\mathsf{A}_{Obs}) \to \mathsf{A}$ is the unique morphism extending the inclusion of A_{Obs} into A (the same for ϵ_B), for every observable $B\Sigma$-computation, t, over A, we have:

$$f_s(\epsilon_\mathsf{A}(t)) = \epsilon_\mathsf{B}(f_s^\#(t)).$$

(Note, again, the theorem giving the unique extensions requires A_{Obs} to be S-indexed, not just Obs-indexed!)

6 The *category of models of* $B\Sigma$, denoted by $Beh(B\Sigma)$ comprises the class of $B\Sigma$-algebras together with the sets of $B\Sigma$-behaviour morphisms.

The isomorphisms in $Beh(B\Sigma)$ exactly express the notion of behaviour equivalence in Meseguer and Goguen (1985), Hennicker and Wirsing (1985), Sannella and Wirsing (1983), and Sannella and Tarlecki (1985). This allows us to extend behaviour equivalence uniformly from algebras to functors: behaviour equivalence of functors is the existence of natural isomorphisms.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

As an example for a $B\Sigma$-behaviour morphism, we again refer to the algebras A, B from the beginning of this section. We have already claimed their behavioural equivalence. This means in particular the existence of a $SET(ELEM)$-behaviour morphism in both directions. We will argue that the family of identities $(id_s : \mathsf{A}_s \to \mathsf{B}_s)_{s \in \{Bool,Elem,Nat\}}$ is a $SET(ELEM)$-behaviour morphism. To see this, we first enumerate the family of observable $B\Sigma$-computations over A:

$$\begin{aligned}
T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Bool} &= \{true, false\} \cup \{true_{\mathsf{A}}, false_{\mathsf{A}}\} \cup \\
&\quad \{is_in(x,s) \mid x \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Elem}, \\
&\qquad s \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Set}\} \\
T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Elem} &= \{a, \ldots, z\} \\
T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Nat} &= \{0\} \cup \{0_{\mathsf{A}}, 1, 2, 3, \ldots\} \\
&\quad \cup \{succ(n) \mid n \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Nat}\} \\
&\quad \cup \{card(s) \mid s \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Set}\} \\
T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Set} &= \{\varnothing\} \\
&\quad \cup \{in(x,s) \mid x \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Elem}, \\
&\qquad s \in T_{SET(ELEM)}(\mathsf{A}_{Obs})_{Set}\}.
\end{aligned}$$

The proof of $f(\epsilon_{\mathsf{A}}(t)) = \epsilon_{\mathsf{B}}(f^{\#}(t))$ for all $t \in T_{SET(ELEM)}(\mathsf{A}_{Obs})$ of observable sort can now be done by structural induction over the structure of t. We want to show just the example $t = card(in(a, in(a, \varnothing)))$:

$$\begin{aligned}
& f(\epsilon_{\mathsf{A}}(t)) &=& \epsilon_{\mathsf{B}}(card(in(a, in(a, \varnothing)))) \\
\iff\ & f(card_{\mathsf{A}}(in_{\mathsf{A}}(a, in_{\mathsf{A}}(a, \varnothing_{\mathsf{A}})))) &=& \epsilon_{\mathsf{B}}(card(in(a, in(a, \varnothing)))) \\
\iff\ & f(card_{\mathsf{A}}(\{a\})) &=& card_{\mathsf{B}}(in_{\mathsf{B}}(a, in_{\mathsf{B}}(a, \varnothing_{\mathsf{B}}))) \\
\iff\ & f(1) &=& card_{\mathsf{B}}(a.a) \\
\iff\ & 1 &=& \text{`number of distinct elements in'}(a.a).
\end{aligned}$$

Finally, we are ready for the definition of the specification frame.

Definition 3.6. (The behaviour equational specification frame *BEQSF*) The *behaviour equational specification frame* $BEQSF = (BSPEC, Beh)$ consists of the category $BSPEC$ as defined in Definition 3.2 and of the functor $Beh : BSPEC^{op} \to CAT$. It maps specifications $B\Sigma$ to model categories $Beh(B\Sigma)$ according to Definition 3.5.
Behaviour specification morphisms $h : B\Sigma_1 \to B\Sigma_2$ are mapped to behaviour forgetful functors $BU_h : Beh(B\Sigma_2) \to Beh(B\Sigma_1)$. They are defined as in the standard equational algebra case.

One remark is necessary regarding the well-definedness of the forgetful functors BU_h in this definition. The mapping of algebras is straightforward. But consider a $B\Sigma$-behaviour morphism $f_2 : \mathsf{A}_2 \to \mathsf{B}_2$ in $Beh(B\Sigma_2)$; the standard way to define $BU_h(f_s) : BU_h(\mathsf{A}_2) \to BU_h(\mathsf{B}_2)$ is

$$BU_h(f_2)_s =_{def} f_{2,h(s)}$$

for all $s \in Obs_1$. Now it is the property of behaviour specification morphisms to preserve

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

the observable sorts (see Definition 3.2), which guarantees $h(s) \in Obs_2$ and thus the existence of $f_{2,h(s)}$.

In the list of examples of Section 2 we mentioned that *BEQSF* seen as the *obvious* institution does not satisfy the *satisfaction condition*. This institution would be constructed by separating the equations into the sentence functor:

$$Beh\_Inst\_? = (BSIG, Beh, Sent, \models_{Beh})$$

with the following components:

1 *BSIG* is the full subcategory of *BSPEC* of specifications without equations.
2 $Beh : BSIG \to CAT^{op}$ is the corresponding restriction of $Beh : BSPEC^{op} \to CAT$ seen as a functor that is contravariant in its codomain.
3 $Sent : BSIG \to Set$ maps every $B\Sigma$ to the maximum set of Σ-equations. Every signature morphism in *BSIG* is mapped to the standard equation translation function as given in *EQSF*.
4 $\models_{Beh,B\Sigma} \subseteq Beh(B\Sigma) \times Sent(B\Sigma)$ is behaviour satisfaction as defined in Definition 3.4.

The satisfaction condition for institutions requires the following: given an arbitrary signature morphism $\varphi : B\Sigma_1 \to B\Sigma_2$ in *BSIG*, an equation $e_1 \in Sent(B\Sigma_1)$ and an algebra $\mathsf{A}_2 \in Beh(B\Sigma_2)$, we have

$$\mathsf{A}_2 \models_{Beh} Sent(\varphi)(e_1) \iff Beh(\varphi)(\mathsf{A}_2) \models_{Beh} e_1.$$

Now consider the following (counter-) example:

$B\Sigma_1$:	**sorts:**	s_1 (non-observable)
	opns:	$a, b : \to s_1$
$B\Sigma_2$:	**sorts:**	s_1 (non-observable)
		s_2 (observable)
	opns:	$a, b : \to s_1$
		$f : s_1 \to s_2$.

Let $\varphi : B\Sigma_1 \to B\Sigma_2$ be the inclusion morphism, the equation e_1 be given by $e_1 \equiv (\varnothing; a = b)$ and the algebra A_2 be given by:

$$\begin{aligned} \mathsf{A}_{2,s_1} &=_{def} \{1, 2\} \\ \mathsf{A}_{2,s_2} &=_{def} \{x, y\} \\ a_{\mathsf{A}_2} &=_{def} 1 \\ b_{\mathsf{A}_2} &=_{def} 2 \\ f_{\mathsf{A}_2} &=_{def} \{1 \mapsto x, 2 \mapsto y\}. \end{aligned}$$

Now, we have $Sent(\varphi)(e_1) = e_1$, and $\mathsf{A}_1 =_{def} Beh(\varphi)(\mathsf{A}_2)$ given by

$$\begin{aligned} \mathsf{A}_{1,s_1} &= \{1, 2\} \\ a_{\mathsf{A}_1} &= 1 \\ b_{\mathsf{A}_1} &= 2. \end{aligned}$$

Thus, $\mathsf{A}_1 \models_{Beh} e_1$ is true because there are no observable contexts to embed $a = b$ in $B\Sigma_1$. But $\mathsf{A}_2 \models_{Beh} Sent(\varphi)(e_1) = e_1$ is false because the observable $B\Sigma_2$-context $f(z)$ yields the requirement $\mathsf{A}_2 \models (\varnothing; f(a) = f(b))$, which is false because of $x \neq y$.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

We have seen that the 'only if' part from right to left in the satisfaction condition is corrupted. We can interpret this phenomenon as the consequence of introducing new observations by the operation f in $B\Sigma_2$. On the one hand, this is methodologically not desired because of the principle of information hiding, which should be preserved. On the other hand, there are many sensible examples in which additional observations are useful and do not cause problems. For example, the extension of a specification by predicates that operate on non-observable types includes operations $p : s \to Bool$ that essentially add new observations. This should not be ruled out.

However, Goguen in Goguen (1989) deals with this kind of restriction, but he seems to avoid the described problems by working in parallel with the standard equational institution.

We can indeed define a proper institution of behaviour specifications when we attach to every equation the set of substitutions and contexts that fix its satisfaction:

$$e \equiv [(Y; l = r),\ Subst_e \subseteq [Y \to T_\Sigma(X_{Obs})],\ Cont_e \subseteq T_\Sigma(X_{Obs})_{Obs}].$$

Satisfaction then restricts Definition 3.4 to the substitutions and contexts in $Subst_e$ or $Cont_e$, respectively. If we extend sentence translation to the substitutions and contexts in the obvious way, we can finally assure the satisfaction condition. But this is just to sketch an idea of how to remedy the deficiency in observing behaviour specifications in an institutional context.

The final goal for us in this section is to fix the class of pushouts PO_{Beh} for which $(BSPEC, PO_{Beh})$ has amalgamations. In order to reach this, we first have to state two technical auxiliary definitions.

Definition 3.7. (Observable consequence, derived equational specification) Given a behaviour specification $B\Sigma = (Obs, \Sigma)$ with $\Sigma = (S, OP, E)$.

1 The set of *observable consequences* of E, denoted by $Obs(E)$, is defined by

$$Obs(E) =_{def} \{(X_{Obs}; l = r) \mid l, r \in T_\Sigma(X_{Obs})_s, s \in Obs; E \vdash (X_{Obs}; l = r)\}$$

Here, $\vdash$ denotes the usual equational calculus relation.

2 The specification *behaviourally derived* from $B\Sigma$, denoted by $Obs(B\Sigma)$, is defined by

$$Obs(B\Sigma) =_{def} (S, OP, Obs(E)).$$

$Obs(B\Sigma)$ is a standard equational specification in $EQSF$.

Before we come to the second technical definition, we will state a proposition that says that the algebra classes of $B\Sigma$ and $Obs(B\Sigma)$ coincide in the sense that every $EQSF$-algebra of $Obs(B\Sigma)$ can be seen as a $BEQSF$-algebra of $B\Sigma$, and *vice versa*. The algebras in these specification frames have the same representations.

Proposition 3.8. Given a behaviour specification $B\Sigma = (Obs, (S, OP, E))$, we have:

$$\mathsf{A} \in Beh(B\Sigma) \iff \mathsf{A} \in Alg(Obs(B\Sigma)).$$

Remember that Alg denotes the model functor of $EQSF$ (see the first example in the list of Section 2).

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Proof.

$\Rightarrow$ $\mathsf{A} \models_{Beh} E$ implies $\mathsf{A} \models_{Beh} Obs(E)$ because $Obs(E)$ is a subset of the equational closure of E. $\mathsf{A} \models_{Beh} Obs(E)$ implies $\mathsf{A} \models Obs(E)$ because every equation $e \in Obs(E)$ is an observable $B\Sigma$-context of itself.

$\Leftarrow$ $\mathsf{A} \models_{Beh} E$ requires the standard satisfaction of observable consequences of E built by substitution and context embedding. This is already given if $\mathsf{A} \models Obs(E)$, the set of *all* observable consequences. □

This proposition has shown that the object classes $Beh(B\Sigma)$ and $Alg(Obs(B\Sigma))$ are equal. However, the definition of homomorphisms in $Alg(Obs(B\Sigma))$ differs from the definition of $B\Sigma$-behaviour morphisms in that the latter ones only provide mappings between the observable carriers.

This implies the existence of a particular *forgetful functor*

$$U_{B\Sigma} : Alg(Obs(B\Sigma)) \to Beh(B\Sigma),$$

which is the identity on algebras and defines for every $h : \mathsf{A} \to \mathsf{B}$ the $B\Sigma$-behaviour morphism $U_{B\Sigma}(h) : \mathsf{A} \to \mathsf{B}$ by $(U_{B\Sigma}(h))_s =_{def} h_s$ for every $s \in Obs$. There are various adjoint constructions to $U_{B\Sigma}$. They are usually called *realizations* (after Goguen and Meseguer (1982)). Some specific realizations within our framework have been studied in Nivela (1987) and Nivela and Orejas (1987).

Definition 3.9. (Observable A-equations) Given a behaviour specification $B\Sigma$ and a $B\Sigma$-algebra A, the set of *observable* A-*equations*, denoted by $obs\_eq(\mathsf{A})$, is defined by

$$obs\_eq(\mathsf{A}) =_{def} \{(l = r) \mid l, r \in T_\Sigma(\mathsf{A}_{Obs})_s, s \in Obs; \epsilon_{\mathsf{A}}(l) = \epsilon_{\mathsf{A}}(r)\},$$

where ϵ_{A} is the same as in Definition 3.5.

In terms of Definition 3.5, $obs\_eq(\mathsf{A})$ contains all pairs of observable $B\Sigma$-computations over A of the same sort whose A-evaluations coincide.

Now we can state the first property of *BEQSF*.

Theorem 3.10. *BEQSF* has free extensions (*cf.* 2.11).

The complete proof of this theorem is too long to be repeated here – it can be found in Nivela and Orejas (1987) and Cornelius (1990b). However, we will give the necessary construction and make a few remarks on it.

Proof. (Sketch) We have to show the existence of an adjunction $BFree_h \dashv Beh(h)$ for any given behaviour specification morphism $h : B\Sigma_1 \to B\Sigma_2$. That is, we have to define $BFree_h : Beh(B\Sigma_1) \to Beh(B\Sigma_2)$. We will apply the well-known property that the free constructions on the objects uniquely extend to the morphisms yielding a free functor. Therefore, we only define for a given $B\Sigma_1$-algebra A_1:

$$BFree_h(\mathsf{A}_1) =_{def} T_{\Sigma_2}(\mathsf{A}_{1,Obs_1})|_{\equiv h(obs\_eq(\mathsf{A}_1)) \cup E_2}.$$

The unit of the adjunction $\eta_{\mathsf{A}_1} : \mathsf{A}_1 \to Beh(h) \circ BFree_h(\mathsf{A}_1)$ is a $B\Sigma_1$-behaviour morphism, that is, to be defined only on the observable carriers. Thus, it is well defined to set for every $s \in Obs_1$

$$\eta_{\mathsf{A}_1,s}(a) =_{def} [a].$$

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

$T_{\Sigma_2}(\mathsf{A}_{1,Obs_1})$ denotes the term algebra that is built over the elements of the observable carriers of A_1 of sort $s \in Obs_1$ seen as additional syntactical constants $a : \rightarrow h(s)$ to extend Σ_2. This term algebra is free by construction.

Additionally, it has to satisfy the equations that identify the *semantical* elements $a \in \mathsf{A}_{1,s}$ with the possible *syntactical* terms generating them. This is done by factorizing with $\equiv_{h(obs\_eq(\mathsf{A}_1))}$. Finally, the algebra has to be at least a $B\Sigma_2$-algebra. That is, it has to satisfy E_2 behaviourally. This is achieved by factoring $T_{\Sigma_2}(\mathsf{A}_{1,Obs_1})$ with $\equiv_{E_2}$. □

Remarks will accompany this theorem:

— When $Obs_1 = S_1$, $BFree_h$ is the standard free construction of *EQSF*.
— When $Obs_1 \neq S_1$, the standard free construction of *EQSF* is, in general, not even a functor. One implication of this is that the *persistency lemma* (see Ehrig and Mahr (1985)) does not generalize to the behavioural case. That is, given a persistent free functor $BFree_h$, assuming h to be injective, there is not necessarily a strongly persistent free functor. The reason is that we do not have direct access to the non-observable carriers. A possible solution to this problem would be a factorization of $Beh(B\Sigma)$ identifying all isomorphic algebras with equal observable parts. We conjecture that in that context for injective specification morphisms it is possible to prove a similar persistency lemma. However, there remain some open questions.
— We already discussed the variations in *BSPEC* resulting from different preservation properties of the observable and the non-observable sorts (Remark 3.3). *Weak* specification morphisms, preserving only the observable sorts, admit equally defined free constructions. The construction of $BFree_h$ relies only on this preservation.

Before we discuss the special class PO_{Beh}, we first show the general existence of pushouts.

Proposition 3.11. *BSPEC* is finitely cocomplete.

Proof. We show finite cocompleteness by proving the existence of initial objects in *BSPEC* and of all pushouts:

1 The empty specification is trivially initial in *BSPEC*.
2 Given two behaviour specification morphisms $f_i : B\Sigma_0 \rightarrow B\Sigma_i$, $i = 1, 2$. Let $(\Sigma_3, g_1 : \Sigma_1 \rightarrow \Sigma_3, g_2 : \Sigma_2 \rightarrow \Sigma_3)$ be a pushout in *EQSF*. Then it is routine to check that

$$B\Sigma_3 =_{def} (g_1(Obs_1) + g_2(Obs_2), \Sigma_3),$$

together with g_1, g_2, interpreted as behaviour specification morphisms, is a pushout in *BSPEC*. □

As a motivation for PO_{Beh}, we want to show by example a pushout in *BSPEC* that does not admit amalgamations and extensions, respectively:

1 Look at the following *BSPEC*-pushout diagram. The involved behaviour specification morphisms are the identities:

$B\Sigma_0$: **sorts:** s_0 (non-observable)
opns: $a, b : \rightarrow s_0$
$B\Sigma_1$: **sorts:** s_0 (non-observable)
opns: $a, b : \rightarrow s_0$
eqns: $a = b$

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

$B\Sigma_2$: **sorts:** s_0 (non-observable)
s_2 (observable)
opns: $a, b : \rightarrow s_0$
$g : s_0 \rightarrow s_2$
$B\Sigma_3$: **sorts:** s_0 (non-observable)
s_2 (observable)
opns: $a, b : \rightarrow s_0$
$g : s_0 \rightarrow s_2$
eqns: $a = b$

This pushout does not admit amalgamations. To see this, consider a $B\Sigma_1$-algebra A_1 with $a_{\mathsf{A}_1} \neq b_{\mathsf{A}_1}$. This is possible because there is no observable $B\Sigma_1$-context to embed the equation $a = b$ in. It therefore does not restrict the model class of the signature of $B\Sigma_1$.

On the other hand, let A_2 be $B\Sigma_2$-algebra with the same forgetful functor image as A_1 (this is a necessary prerequisite for amalgamation). That means, in particular, that $a_{\mathsf{A}_2} \neq b_{\mathsf{A}_2}$ again. Let g_{A_2} be injective. Then $g_{\mathsf{A}_2}(a_{\mathsf{A}_2}) \neq g_{\mathsf{A}_2}(b_{\mathsf{A}_2})$, implying that $a = b$ does not hold behaviourally in A_2. This means that there is no $B\Sigma_3$-algebra whose forgetful functor image with respect to $B\Sigma_2$ coincides with A_2. But this would have been the case for amalgamation.

2 The same pushout diagram does not have extensions. The following arguments already induce the non-existence of amalgamations due to Theorem 2.12. But we hope to give some more insight into the behavioural theory by explaining both reasons in detail.

The model classes of $B\Sigma_0$ and $B\Sigma_1$ coincide because there are no observable sorts. Moreover, every two algebras in this model class are behaviourally equivalent, because the only $B\Sigma_0$-behaviour morphism between them is the empty family of mappings, which is necessarily an isomorphism. As a consequence, the identity functor $id : Beh(B\Sigma_0) \rightarrow Beh(B\Sigma_1)$ is trivially strongly persistent. For the same reason it is also a left adjoint.

On the other hand, there is no strongly persistent functor $F : Beh(B\Sigma_2) \rightarrow Beh(B\Sigma_3)$ because no $B\Sigma_2$-algebra A_2 with $a_{\mathsf{A}_2} \neq b_{\mathsf{A}_2}$ and g_{A_2} injective, can be preserved.

We will characterize PO_{Beh} by the following *observation preserving property*. A PO-diagram has this property if in $B\Sigma_3$ the ways to *observe* the terms of non-observable sort remain the same as the ways given in $B\Sigma_1$ and $B\Sigma_2$. The mixture of operations of $B\Sigma_1$ and $B\Sigma_2$ in $B\Sigma_3$ potentially yields such new observations. This can either include observable terms in $B\Sigma_3$ that aren't terms in either $B\Sigma_1$ nor $B\Sigma_2$, or it can include the even more likely case that $B\Sigma_1$ doesn't know the observations possible in $B\Sigma_2$, but that their meeting in $B\Sigma_3$ causes semantical problems. This last problem will be made clear in the following paragraph.

In the example above, we have the non-observable sort s_0 in every specification. The equation $a = b$ in $B\Sigma_1$ states a requirement relative to all *existing* observable computations that include the equation. But in $B\Sigma_2$ a new possibility to observe the sort s_0 is given by the operation g. This leads to the conflict in $B\Sigma_3$ where the equation and the operation

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

g meet. Thus, there is no algebra in which a and b are differently interpreted if g is an injective mapping.

Definition 3.12. (Observation preserving property) Assume a *BSPEC*-pushout diagram as in Figure 11. Let X_{Obs_3} be the family of sets of variables of the observable sorts in $B\Sigma_3$.

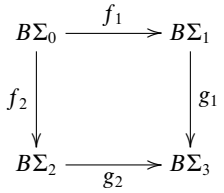


Fig. 11. *BSPEC*-pushout diagram

1 An observable $B\Sigma_3$-computation $t_3 \in T_{\Sigma_3}(X_{Obs_3})$ is said to be *clean* with respect to a given pushout diagram iff no sort of any of its non-observable sub-terms has its origin in $B\Sigma_0$, that is, for every non-observable sort s of a sub-term of t_3 we have: $s \notin g_1 \circ f_1(S_0)$.
2 A pushout diagram satisfies the *observation preserving property* iff for every observable $B\Sigma_3$-computation t_3 there is a clean $B\Sigma_3$-computation t_3' that is equivalent with respect to the equations in $B\Sigma_3$, that is, $B\Sigma_3 \vdash (X_{Obs_3}; t_3 = t_3')$.

There is the legitimate question as to whether the class of pushouts satisfying the observation preserving property is not too restrictive in the sense of lacking practical applicability. More specifically, the decision for a sub-class of pushouts leads to a trade-off between practical relevance on the one hand, and the provision of the necessary theoretical results on the other hand.

Fortunately there is a large number of standard cases where the observation preserving property holds. First, it holds whenever the signature $B\Sigma_0$ is completely observable. This situation is very frequent because $B\Sigma_0$ typically plays the role of a parameter or a part common to $B\Sigma_1$ and $B\Sigma_2$. In both cases it plays a *public* role: a formal parameter should express conceptual *openness*, and the designation of common parts is similar. This does not match the general intuition that non-observable parts are mostly specialized and locally defined.

Second, even in the case of a non-observable type in $B\Sigma_0$ one generally expects this type to be defined completely within $B\Sigma_0$. In other words, extensions in $B\Sigma_1$ and $B\Sigma_2$ are expected to rely only on the observations originally provided in $B\Sigma_0$. And in that case, where every new (say $B\Sigma_1$) observation $f_1(c)$ of a non-observable term c is equivalent to a $B\Sigma_0$-observation $f_0(c)$, the observation preserving property depends only on the completeness of the specification of the non-observable type in $B\Sigma_0$. As this kind of completeness, namely the existence of a completely observable c' with $B\Sigma_0 \vdash f_0(c) = c'$, is one suitable correctness property in the framework of constructive specifications with initial semantics, the additional requirement of satisfaction of the observation preserving property should generally be fulfilled.

In the example above, the observation preserving property does not hold. For example the only observable $B\Sigma_3$-computations are $g(a)$ and $g(b)$. Both contain a sub-term of sort s_0 originating from $B\Sigma_0$.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

The first of the facts in Lemma 3.14 will justify the name of the property. It states the consequence of the observation preserving property that no *new* observations are produced when constructing the pushout, that is, the *old* observations are preserved.

In the case of weak specification morphisms we can apply the same definition. It also yields the class of pushouts that admit amalgamations and, thus, extensions.

Now, PO_{Beh} will denote the class of $BSPEC$-pushouts that satisfy the observation preserving property.

Remark 3.13. (Extension of specification morphisms to (ground) terms) Before we come to some important facts about the pushouts in PO_{Beh}, we need to state some relations between specification morphisms and (ground-) terms of the signature: a (behaviour-) specification morphism $\varphi : B\Sigma_1 \to B\Sigma_2$ induces canonically a mapping $gterm(\varphi) : T_{\Sigma_1} \to T_{\Sigma_2}$ between the ground terms of the specifications.
We want to expand this mapping to the sets of terms built over variables X_{Σ_1} and X_{Σ_2}. In order to do so, we assume, without loss of generality, for every $s_2 \in \varphi(S_1)$ (that is, sort of X_{Σ_2}) that is in the image of φ satisfies the following property:

$$\biguplus\{X_{\Sigma_1,s_1} \mid \varphi(s_1) = s_2\} \subseteq X_{\Sigma_2,s_2}.$$

This property enables us for every term $t \in T_{\Sigma_1}(X_{\Sigma_1})$ to take the identity on the variables in t when translating t with the aid of φ.
We do not lose the generality because the 'names' of the variables, the symbols, do not have any semantical significance in themselves. The fact that we have this property means we can easily extend $gterm(\varphi)$ to a mapping $term(\varphi) : T_{\Sigma_1}(X_{\Sigma_1}) \to T_{\Sigma_2}(X_{\Sigma_2})$, which acts like $gterm(\varphi)$ but, additionally, is the identity on the variables.

The following lemma states some technical results concerning pushouts in PO_{Beh}. The first fact essentially says that every observable $B\Sigma_3$-computation is already a $B\Sigma_1$- or a $B\Sigma_2$-computation. The second fact says that the set of valid equations between observable consequences in $B\Sigma_3$ can be deduced by using the sets of equations in $B\Sigma_1$ and $B\Sigma_2$, respectively. The third fact states a sufficient condition for the satisfaction of the observation preserving property. It says that it is enough to check the *minimal* observable $B\Sigma_3$-computations for the property of Definition 3.12.

Lemma 3.14. Given a pushout diagram in PO_{Beh} as depicted in Figure 11, let X_{Obs_i}, $i = 0, \ldots, 3$ be families of sets of variables according to the convention of the above Remark 3.13:

1 Given a clean observation $t_3 \in T_{\Sigma_3}(X_{Obs_3})$, for every $t_3' \in Min(t_3)$ (see Definition 3.5) we have

$$t_3' \in term(g_2)(T_{\Sigma_2}(X_{Obs_2})) \cup term(g_1)(T_{\Sigma_1}(X_{Obs_1})).$$

2 If we let $Obs(E_i)$ denote the sets of equations (observable consequences) in the behaviourally derived specifications $Obs(B\Sigma_i)$ $(i = 0, \ldots, 3)$ according to Definition 3.7, then for every pair of observable $B\Sigma_3$-computations $l, r \in T_{\Sigma_3}(X_{Obs_3})$, we have

$$Obs(E_3) \vdash (X_{Obs_3}; l = r) \Leftrightarrow term(g_1)(Obs(E_1)) \cup term(g_2)(Obs(E_2)) \vdash (X_{Obs_3}; l = r)$$

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

In particular, we have that the $Obs(B\Sigma_i)$-diagram of behaviourally derived specifications is a pushout in *EQSF*.

3 If for every *minimal* observable $B\Sigma_3$-computation $t_3 \in T_{\Sigma_3}(X_{Obs_3})$ there is a clean observation $t'_3 \in T_{\Sigma_3}(X_{Obs_3})$ such that $E_3 \vdash (X_{Obs_3}; t_3 = t'_3)$, then the pushout diagram satisfies the observation preserving property.

The proof of the facts of this lemma can be found in Orejas *et al.* (1989) and Cornelius (1990a). We omit them because of their mere technical character.

The following theorem will complete this section.

Theorem 3.15. $(BEQSF, PO_{Beh})$ has amalgamations and extensions.

Proof. As amalgamations imply extensions (Theorem 2.12), we can restrict ourselves to showing the amalgamation property.

Let $\mathsf{A}_i \in Beh(B\Sigma_i)$, $i = 1, 2$, be such that $BU_{f_1}(\mathsf{A}_1) = BU_{f_2}(\mathsf{A}_2)$. From Proposition 3.8, we can (in *EQSF*) define

$$\mathsf{A}_3 = \mathsf{A}_1 +_{(f_1,f_2)} \mathsf{A}_2.$$

As *EQSF* has amalgamations for the class of all pushouts, we already know that A_3 is unique with respect to $BU_{g_1}(\mathsf{A}_3) = \mathsf{A}_1 \ \wedge \ BU_{g_2}(\mathsf{A}_3) = \mathsf{A}_2$, but we still need to show that $\mathsf{A}_3 \in Beh(B\Sigma_3)$, that is:

$$\begin{array}{lllll} & \mathsf{A}_3 & \models_{Beh} & E_3 \\ \Leftrightarrow & \mathsf{A}_3 & \models & Obs(E_3) \\ \Leftrightarrow & \mathsf{A}_3 & \models & term(g_1)(Obs(E_1)) \cup term(g_2)(Obs(E_2)). \end{array}$$

It remains for us to show the existence of amalgamated sums of $B\Sigma$-behaviour morphisms. Let $h_i : A_i \to A'_i \in Beh(B\Sigma_i)$, $i = 1, 2$, be such that $BU_{f_1}(h_1) = BU_{f_2}(h_2)$. We define $h_3 : \mathsf{A}_1 +_{(f_1,f_2)} \mathsf{A}_2 \to \mathsf{A}'_1 +_{(f_1,f_2)} \mathsf{A}'_2$ for all $s \in Obs_3$ by

$$h_{3,s} =_{def} \begin{cases} h_{1,g_1^{-1}(s)} & ; \quad s \in g_1(S_1) \\ h_{2,g_2^{-1}(s)} & ; \quad s \in g_2(S_2). \end{cases}$$

h_3 is a well-defined $B\Sigma_3$-behaviour morphism. Now let $t_3 \in T_{\Sigma_3}(\mathsf{A}_3)$ be an observable $B\Sigma_3$-computation over A_3. We need to show

$$h_3 \circ \epsilon_3(t_3) = \epsilon'_3 \circ h_3^{\#}(t_3). \tag{1}$$

Because of the observation preserving property, we find a clean observation m_3 such that $B\Sigma_3 \vdash (\mathsf{A}_{3,Obs};\ t_3 = m_3)^{\dagger}$. That is, we can equivalently replace t_3 by m_3 in Equation 1. We proceed with sub-term induction:

— **base** If m_3 is minimal, we have $m_3 \in term(g_1)(T_{\Sigma_1}(\mathsf{A}_{1,Obs})) \cup term(g_2)(T_{\Sigma_2}(\mathsf{A}_{2,Obs}))$. That is, we can apply the commutation property of h_1 or h_2, respectively.

— **step** If m_3 is not minimal, we have that there is $m'_3 \in Min(m_3)$ and a substitution σ for the (fresh) variables[‡] in m'_3 with terms in $T_{\Sigma_3}(\mathsf{A}_{3,Obs})$ such that $\sigma^*(m'_3) = m_3$.

[†] We can interpret $\mathsf{A}_{3,Obs}$ as a particular family of *variables*

[‡] That is, the variables inserted when building the set $Min(m_3)$!

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

For every (fresh) x in m_3', $\sigma(x)$ is a sub-term of m_3 and an observable $B\Sigma_3$-computation, too. That is, applying the induction hypothesis yields commutation for every $\sigma(x)$. Additionally, h_3 commutes with respect to m_3' because it is minimal.

This completes the proof. □

4. Abstract module specifications

Elements of the theory of *algebraic* module specifications were developed in Ehrig and Weber (1985), Ehrig and Weber (1986), Weber and Ehrig (1986), and Blum *et al.* (1987 and, later, a comprehensive account was presented in Ehrig and Mahr (1990). The basis for this theory are equational algebraic specifications, that is, the specification frame *EQSF*. In this section we carry out the generalization to specification frames in the sense of Section 2. We present the relevant definitions for abstract module specifications, including the basic interconnection mechanisms. They serve to construct new and bigger modules from smaller ones. The main theorems of this section state the correctness and compositionality of these operations.

Definition 4.1. (Abstract module specification) Let $SF = (ASPEC, Catmod)$ be a specification frame:

1 An (abstract) *SF-module specification* $AMOD = (e, s, i, v)$ is a commutative diagram in $ASPEC$ as depicted in Figure 12. We call $A\Sigma_{PAR}$ the (abstract) parameter specification, $A\Sigma_{EXP}$ the (abstract) export interface specification, $A\Sigma_{IMP}$ the (abstract) import interface specification and $A\Sigma_{BOD}$ the (abstract) body specification.

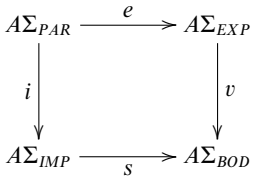


Fig. 12. Abstract module specification *AMOD*

2 The *functorial semantics* of *AMOD* is defined to be the following class of functors:

$$Sem(AMOD) =_{def} \{V_v \circ F_s : Catmod(A\Sigma_{IMP}) \to Catmod(A\Sigma_{EXP}) \mid F_s \dashv V_s\}.$$

3 *AMOD* is called *internally correct* if there exists a strongly persistent F_s with $F_s \dashv V_s$. In this case we call $V_v \circ F_s \in Sem(AMOD)$ a *standard semantic functor* for *AMOD*.

Remark 4.2. (Abstract parameterized specification) An abstract module specification with $e = s$, $i = id_{A\Sigma_{PAR}}$ and $v = id_{A\Sigma_{EXP}}$, is an abstract parameterized specification with initial semantics in the sense of Ehrig and Mahr (1985) and Ehrig and Große-Rhode (1994).

The semantics $Sem(AMOD)$ of an abstract module specification is an isomorphism class of functors, each transforming import to export data types. In the case of internal correctness, the import data type is completely protected by the functors in $Sem(AMOD)$. Other notions of correctness can be found in Ehrig and Mahr (1990).

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

The following definition allows us to construct the category of abstract module specifications.

Definition 4.3. (Abstract module specification morphism) Assume we are given two *SF*-module specifications $AMOD_i$, $i = 1, 2$, where these indices are assumed to be propagated to the components of the modules. Then an *SF-module specification morphism* $\varphi : AMOD_1 \to AMOD_2$ consists of a 4-tuple of *ASPEC*-morphisms $\varphi = (\varphi_P, \varphi_E, \varphi_I, \varphi_B)$ such that every square of the diagram in Figure 13 commutes. The composition of *SF*-module specification

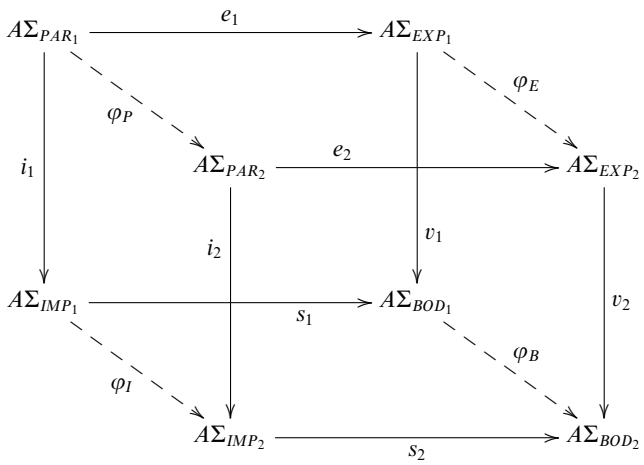


Fig. 13. (Abstract) module specification morphism $\varphi : AMOD_1 \to AMOD_2$

morphisms is defined component-wise, and the category of *SF*-module specifications and -morphisms is denoted by *Modcat*(*SF*).

The next step is to study operations on module specifications, especially composition, union and actualization. Composition describes a *vertical* plugging of modules: the data types required in the import interface of a *higher* module specification are provided by a *lower* module specification in its export interface. This relationship is expressed by a specification morphism $h_1 : A\Sigma_{IMP_1} \to A\Sigma_{EXP_2}$.

The formal parameter specification of $AMOD_1$ becomes the parameter of the ensuing module specification and has, therefore, to be mapped to the parameter specification of the *lower* module $AMOD_2$. This is achieved by a specification morphism $h_2 : A\Sigma_{PAR_1} \to A\Sigma_{PAR_2}$ with $h_1 \circ i_1 = e_2 \circ h_2$ that is compatible with h_1.

Each of the following operations involves the specification of a diagram, parts of which are the argument and the result module specifications. Moreover, in each case certain sub-diagrams have to be pushouts so as to allow us to make use of amalgamation and/or extension to prove correctness and compositionality. In each definition we therefore assume a specification frame with pushouts (SF, PO). The correctness and compositionality theorems will then be based on the additional assumption that (SF, PO) has amalgamations.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Definition 4.4. (Composition of abstract module specifications) Let a specification frame (SF, PO) with pushouts, SF-module specifications $AMOD_i$, $i = 1, 2$ (where these indices apply to the components of $AMOD_i$, $i = 1, 2$ in the expected way), and specification morphisms in SF

$$h_1 : A\Sigma_{IMP_1} \to A\Sigma_{EXP_2} \text{ and}$$
$$h_2 : A\Sigma_{PAR_1} \to A\Sigma_{PAR_2}$$

be given. Then the *composition* of $AMOD_1$ and $AMOD_2$ via $h =_{def} (h_1, h_2)$, written $AMOD_1 \circ_h AMOD_2$, is defined by

$$AMOD_1 \circ_h AMOD_2 =_{def} (e_1, \widehat{s} \circ s_2, i_2 \circ h_2, \widehat{v} \circ v_1)$$

if $h_1 \circ i_1 = e_2 \circ h_2$ and if $(\widehat{v}, \widehat{s})$ is a pushout of $(s_1, v_2 \circ h_1)$ in PO. The overall situation is depicted in Figure 14.

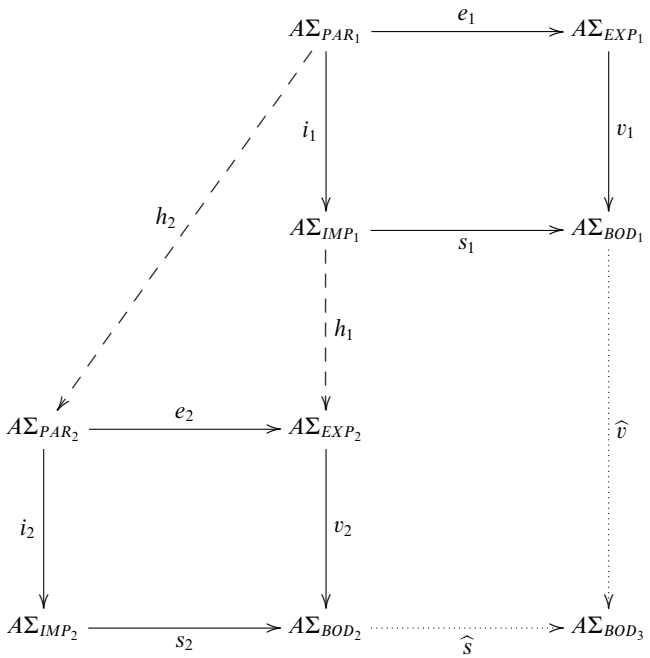


Fig. 14. Composition of $AMOD_1$ and $AMOD_2$

Note that, according to our intuition, the ensuing module $AMOD_1 \circ_h AMOD_2$ inherits the export interface of the higher module $AMOD_1$ and the import of the lower one $AMOD_2$. Note also that there is always an isomorphism class of possible body specifications $A\Sigma_{BOD_3}$. Therefore, to make the composition of abstract module specifications functional, one could introduce a system of representatives for the isomorphism classes of abstract specifications. Another method, which can be applied whenever the abstract theory has been instantiated to some concrete specification frame, is of course to construct pushouts.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Next, we prove the correctness and compositionality of composition under the assumption of internally correct upper and lower modules.

Theorem 4.5. (Correctness and compositionality of composition) Whenever, in Definition 4.4, (SF, PO) has free extensions and $AMOD_i$ is internally correct with strongly persistent functor $F_{s_i} \dashv V_{s_i}$ for $i = 1, 2$, then we have (setting $AMOD_3$ for $AMOD_1 \circ_h AMOD_2$):

1 **Correctness**
$AMOD_3$ is internally correct.

2 **Compositionality**
Let $Sem_i =_{def} V_{v_i} \circ F_{s_i}$ be a standard semantic functor for $AMOD_i$, $i = 1, 2$. Then we can construct a standard semantic functor Sem_3 for $AMOD_3$ by

$$Sem_3 =_{def} Sem_1 \circ V_{h_1} \circ Sem_2.$$

Proof. For Part (1), we have to show the existence of a strongly persistent functor F with $F \dashv V_{\hat{s} \circ s_2}$. To this end, let us note that we can extend F_{s_1} to a strongly persistent $F_{\hat{s}}$ with $F_{\hat{s}} \dashv V_{\hat{s}}$ and $F_{s_1} \circ V_{v_2 \circ h_1} = V_{\hat{v}} \circ F_{\hat{s}}$ (Theorem 2.12). Let $F =_{def} F_{\hat{s}} \circ F_{s_2}$. By Remark 2.4, F is a strongly persistent and left adjoint to $V_{\hat{s} \circ s_2}$.

For Part (2), we show that Sem_3 is the same as $V_{\hat{v} \circ v_1} \circ F$, reasoning in the following way:

$$\begin{aligned} Sem_1 \circ V_{h_1} \circ Sem_2 &= V_{v_1} \circ F_{s_1} \circ V_{h_1} \circ V_{v_2} \circ F_{s_2} \\ &= V_{v_1} \circ F_{s_1} \circ V_{v_2 \circ h_1} \circ F_{s_2} \\ &= V_{v_1} \circ V_{\hat{v}} \circ F_{\hat{s}} \circ F_{s_2} \quad \text{since } F_{s_1} \circ V_{v_2 \circ h_1} = V_{\hat{v}} \circ F_{\hat{s}} \text{ (see above)} \\ &= V_{\hat{v} \circ v_1} \circ F. \end{aligned}$$

□

The next operation to be considered, *union*, again makes use of pushouts.

Definition 4.6. (Union of abstract module specifications) Let (SF, PO) be a specification frame with pushouts. Given three SF-module specifications $AMOD_i$, $i = 0, 1, 2$, and two SF-module specification morphisms $f_i : AMOD_0 \to AMOD_i$, $i = 1, 2$, the *union* of $AMOD_1$ and $AMOD_2$ via f_1, f_2, written $AMOD_1 \oplus_{(f_1, f_2)} AMOD_2$, is defined by

$$AMOD_1 \oplus_{(f_1, f_2)} AMOD_2 =_{def} (e_3, s_3, i_3, v_3),$$

where these four morphisms are uniquely induced by pushouts (g_1^X, g_2^X) of (f_1^X, f_2^X) for $X \in \{PAR, EXP, IMP, BOD\}$ provided that all these pushouts are elements of PO. (See Figure 15 where, for reasons of clarity, we have omitted most of the arrow labels in the module specification morphisms.)

Note that (g_1, g_2) is a pushout of (f_1, f_2) in $Modcat(SF)$.

Theorem 4.7. (Correctness and compositionality of union) Whenever, in Definition 4.6, (SF, PO) has amalgamations, $AMOD_i$ is internally correct with strongly persistent functor $F_{s_i} \dashv V_{s_i}$ for $i = 1, 2$ and the compatibility

$$V_{f_1^{BOD}} \circ F_{s_1} \circ V_{g_1^{IMP}} = V_{f_2^{BOD}} \circ F_{s_2} \circ V_{g_2^{IMP}},$$

holds, we have (setting $AMOD_3$ for $AMOD_1 \oplus_{(f_1, f_2)} AMOD_2$):

1 **Correctness**
$AMOD_3$ is internally correct.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

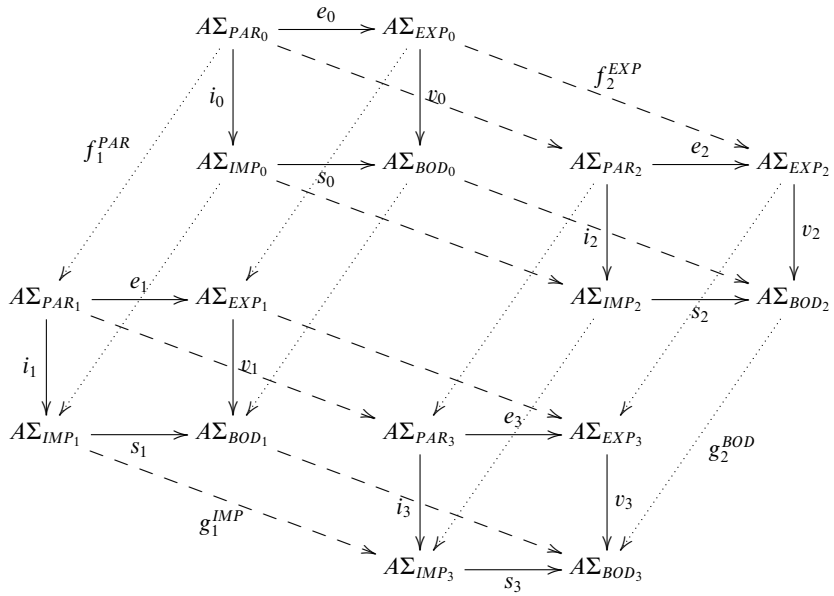


Fig. 15. Union of $AMOD_1$ and $AMOD_2$

2 **Compositionality**

Let $Sem_i =_{def} V_{v_i} \circ F_{s_i}$ be standard semantic functors for $AMOD_i$, $i = 1, 2$. Then we can construct a standard semantic functor Sem_3 for $AMOD_3$ by amalgamating Sem_1 and Sem_2:

$$Sem_3 =_{def} Sem_1 +_{(f_1^{EXP}, f_2^{EXP})} Sem_2$$

Proof.

1 This part is an easy matter of applying Lemma 2.10 to the import and body pushouts, where (s_0, s_1, s_2, s_3) plays the role of the quadruple of morphisms between them.

2 First, the amalgamation of Sem_1 and Sem_2 via f_1^{EXP}, f_2^{EXP} is well defined because Sem_1 and Sem_2 are compatible as follows (*cf.* Lemma 2.10):

$$\begin{aligned} V_{f_1^{EXP}} \circ Sem_1 \circ V_{g_1^{IMP}} &= V_{f_1^{EXP}} \circ V_{v_1} \circ F_{s_1} \circ V_{g_1^{IMP}} \\ &= V_{v_0} \circ V_{f_1^{BOD}} \circ F_{s_1} \circ V_{g_1^{IMP}} \\ &\qquad \text{since } v_1 \circ f_1^{EXP} = f_1^{BOD} \circ v_0 \\ &= V_{v_0} \circ V_{f_2^{BOD}} \circ F_{s_2} \circ V_{g_2^{IMP}} \\ &\qquad \text{since } V_{f_1^{BOD}} \circ F_{s_1} \circ V_{g_1^{IMP}} = V_{f_2^{BOD}} \circ F_{s_2} \circ V_{g_2^{IMP}} \\ &= V_{f_2^{EXP}} \circ Sem_2 \circ V_{g_2^{IMP}} \\ &\qquad \text{analogously to the first three steps.} \end{aligned}$$

Thus, it remains to show that Sem_3 is a standard semantic functor, that is,

$$Sem_3 = V_{v_3} \circ F_{s_3}$$

for some strongly persistent $F_{s_3} : Catmod(A\Sigma_{IMP_3}) \to Catmod(A\Sigma_{IMP_3})$ with $F_{s_3} \dashv V_{s_3}$.

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

To this end, we state $F_{s_3} =_{def} F_{s_1} +_{(f_1^{BOD}, f_2^{BOD})} F_{s_2}$, which is actually the functor used to show (1). All that is left, therefore, is to show $Sem_3 = V_{v_3} \circ F_{s_3}$. We do so by exploiting the fact that Sem_3, as the amalgamated sum of Sem_1 and Sem_2 via f_1^{EXP}, f_2^{EXP}, is unique with respect to $V_{g_i^{EXP}} \circ Sem_3 = Sem_i \circ V_{g_i^{IMP}}$, $i = 1, 2$ (again Lemma 2.10). That is, we show

$$V_{g_i^{EXP}} \circ V_{v_3} \circ F_{s_3} = Sem_i \circ V_{g_i^{IMP}},\ i = 1, 2.$$

First of all, Lemma 2.10 implies that F_{s_3}, as the amalgamated sum of F_{s_1} and F_{s_2} via f_1^{BOD}, f_2^{BOD}, satisfies (and is of course also unique with respect to)

$$V_{g_i^{BOD}} \circ F_{s_3} = F_{s_i} \circ V_{g_i^{IMP}},\ i = 1, 2.$$

Hence,

$$\begin{aligned} V_{g_i^{EXP}} \circ V_{v_3} \circ F_{s_3} &= V_{v_i} \circ V_{g_i^{BOD}} \circ F_{s_3} \quad \text{since } v_3 \circ g_i^{EXP} = g_i^{BOD} \circ v_i \\ &= V_{v_i} \circ F_{s_i} \circ V_{g_i^{IMP}} \\ &= Sem_i \circ V_{g_i^{IMP}}. \end{aligned}$$

□

We conclude this section with the actualization operation, which allows one to instantiate the parameter of an abstract module specification. The instantiation target has to be the body of an abstract *parameterized* specification (*cf.* Remark 4.2). This specification's own parameter becomes the parameter of the new module.

Definition 4.8. (Actualization of abstract module specifications) Let (SF, PO) be a specification frame with pushouts. Given an SF-module specification $AMOD$, an SF-parameterized specification $P\Sigma = (p : P\Sigma_{PAR} \to P\Sigma_{BOD})$ and a *parameter passing morphism* $h : A\Sigma_{PAR} \to P\Sigma_{BOD}$, the *actualization* of $AMOD$ by $P\Sigma$ via h, written $AMOD_h(P\Sigma)$, is described on the basis of Figure 16 by

$$AMOD_h(P\Sigma) =_{def} (e_0 \circ p, s_0, i_0 \circ p, v_0).$$

As a side condition, all squares in Figure 16 that are bordered by at least one dashed arrow must be in PO.

Remark 4.9. (Constructing $AMOD_h(P\Sigma)$) The side condition in Definition 4.8 is such that it is not immediately clear how one could *construct* the actualization diagram in any concrete syntactic category. It is, however, clear that one can add three pushouts to the *initial* diagram straight away so long as they belong to PO. More specifically, one begins with the top and left pushouts and continues with either the bottom or the right one. These two possibilities are entirely symmetrical to each other, so consider choosing the right pushout. What is left after it has been added is to close the diagram by inserting an appropriate s_0. To this end, one can use the universal property of the left pushout so as to obtain an s_0 that makes the diagram commute in the first place. This property already entails that the composition of the left and bottom squares is a pushout. Showing the pushout property of the bottom square in isolation is then a matter of straightforward diagram chasing.

An important additional observation is that the three constructed pushouts have amalgamations, then the derived one has them also. This property is a simple consequence

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

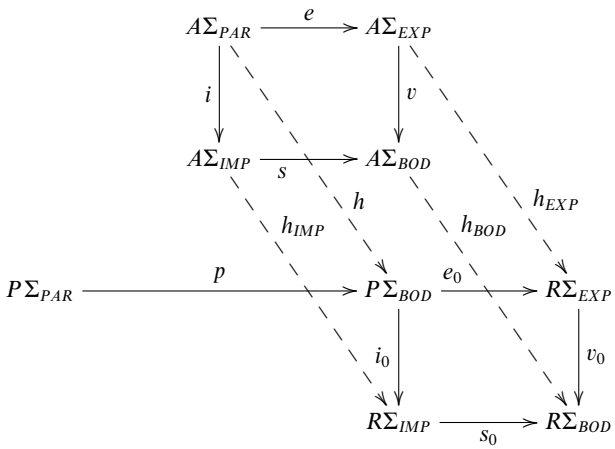


Fig. 16. Actualization of $AMOD$ by $P\Sigma$ via h

of the fact that *Catmod* maps the actualization diagram to a commutative diagram in CAT where, by assumption, the images of the three constructed pushouts are pullbacks. In consequence, whenever PO is the class of all pushouts in the syntactic category whose *Catmod*-image is a pullback, the derived pushout belongs to PO.

If $P\Sigma$ is a *flat specification* with an empty parameter, the ensuing module specification is completely actualized. In other words, it assumes the role of an *abstract interface specification* with constructive body part. For further details, see Ehrig and Mahr (1990).

Theorem 4.10. (Correctness and compositionality of actualization) Whenever, in Definition 4.8, (SF, PO) has amalgamations and $AMOD$ is internally correct with strongly persistent functor $F_s \dashv V_s$, we have:

1 **Correctness**
$AMOD_h(P\Sigma)$ is internally correct.

2 **Compositionality**
Let $Sem =_{def} V_v \circ F_s$ be a standard semantic functor for $AMOD$. Then we can construct a standard semantic functor Sem_R for $AMOD_R$ by amalgamating Sem and $Id_{Catmod(PS_{BOD})}$:

$$Sem_R = Sem +_{(e,h)} Id_{Catmod(P\Sigma_{BOD})}.$$

Proof.

1 By Remark 4.9, the bottom pushout of the actualization diagram has amalgamations. We can thus apply Theorem 2.12 so as to extend F_s to the functor $Ext_{h_{IMP}}(F_s)$: $Catmod(R\Sigma_{IMP}) \to Catmod(R\Sigma_{BOD})$, which is strongly persistent and left adjoint to V_{s_0}. So $AMOD_h(P\Sigma)$ is internally correct.

2 First, the amalgamation of Sem and $Id_{Catmod(P\Sigma_{BOD})}$ via e and h is well-defined because

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

Sem and $Id_{Catmod(P\Sigma_{BOD})}$ are compatible as follows (*cf.* Lemma 2.10):

$$\begin{aligned} V_e \circ Sem \circ V_{h_{IMP}} &= V_e \circ V_v \circ F_s \circ V_{h_{IMP}} \\ &= V_i \circ V_s \circ F_s \circ V_{h_{IMP}} \\ &= V_i \circ V_{h_{IMP}} \quad \text{by strong persistency of } F_s \\ &= V_h \circ V_{i_0} \\ &= V_h \circ Id_{Catmod(P\Sigma_{BOD})} \circ V_{i_0}. \end{aligned}$$

Thus, it remains to show that Sem_R is a standard semantic functor. We do so by proving $Sem_R = V_{v_0} \circ Ext_{h_{IMP}}(F_s)$, exploiting the uniqueness of Sem_R, as the amalgamated sum of *Sem* and $Id_{Catmod(P\Sigma_{BOD})}$ via e, h, with respect to

(a) $V_{h_{EXP}} \circ Sem_R = Sem \circ V_{h_{IMP}}$ and

(b) $V_{e_0} \circ Sem_R = Id_{Catmod(P\Sigma_{BOD})} \circ V_{i_0}$

(once again using Lemma 2.10). That is, we show (a) and (b) with Sem_R replaced by $V_{v_0} \circ Ext_{h_{IMP}}(F_s)$. As for (a):

$$\begin{aligned} V_{h_{EXP}} \circ V_{v_0} \circ Ext_{h_{IMP}}(F_s) &= V_v \circ V_{h_{BOD}} \circ Ext_{h_{IMP}}(F_s) \\ &= V_v \circ F_s \circ V_{h_{IMP}} \quad \text{by Theorem 2.12} \\ &= Sem \circ V_{h_{IMP}}. \end{aligned}$$

As for (b):

$$\begin{aligned} V_{e_0} \circ V_{v_0} \circ Ext_{h_{IMP}}(F_s) &= V_{i_0} \circ V_{s_0} \circ Ext_{h_{IMP}}(F_s) \\ &= V_{i_0} \quad \text{by strong persistency of } Ext_{h_{IMP}}(F_s) \\ &= Id_{Catmod(P\Sigma_{BOD})} \circ V_{i_0}. \end{aligned}$$

Note that our use of Lemma 2.10 is such that we cannot fully embed the diagram associated with this result (bottom half of Figure 6) in the actualization diagram. This aspect is, however, not a problem because what we need from the lemma can already be proved on the basis of what is available here. □

We conclude this section with some remarks on the generalization of the remaining module concepts mentioned in Ehrig and Mahr (1990) in the context of abstract module specifications.

Remark 4.11. (Further concepts and results for abstract module specifications)

1 The distributive laws in section *3D* of Ehrig and Mahr (1990) concerning the compatibility of composition, union and actualization remain valid in arbitrary specification frames with pushouts. The reason is that these results are mainly based on the commutativity of different kinds of pushouts on the syntactic level.

2 More general operations on abstract module specifications, including renaming, partial composition, recursion, product and iteration, can be introduced if we have more general colimit constructions on the syntactic level. The compatibility results between different operations, as shown in Chapter 4 of Ehrig and Mahr (1990), are also a consequence of different kinds of colimits. On the semantic level it seems reasonable to not only require amalgamation and extension for specific pushouts but also for the corresponding colimit constructions (see Claßen (1993)).

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

3 The constructions and results concerning refinement, interface specifications, realizations, development categories, simulation and transformation in Chapters 5 and 6 of Ehrig and Mahr (1990) are mainly based on the existence of pushouts and other colimits on the syntactic level and the existence of the semantics of the component module specifications. This generalizes immediately to abstract module specifications.
4 The generalization of restriction semantics and module specifications with constraints as given in Ehrig and Mahr (1990) needs a more careful analysis and goes beyond the scope of this paper. For restriction constructions in specification frames we refer to Ehrig and Große-Rhode (1994). Some ideas on how to handle constraints are already given in Ehrig *et al.* (1991a).

5. Behaviour module specifications

This section is about the instantiation of the theory of abstract module specifications to behaviour specification frames. First, we consider the instantiation to $(BEQSF, PO_{Beh})$, the specification frame of equational behaviour specifications together with the class of pushouts in the underlying syntactic category $BSPEC$ that satisfy the observation-preserving property: in the ensuing theory, a module is understood to be a commutative diagram of the form

$$\begin{array}{ccc} B\Sigma_{PAR} & \xrightarrow{e} & B\Sigma_{EXP} \\ \downarrow{\scriptstyle i} & & \downarrow{\scriptstyle v} \\ B\Sigma_{IMP} & \xrightarrow[s]{} & B\Sigma_{BOD} \end{array}$$

in $BSPEC$. Its semantics and internal correctness are defined on the basis of the behaviour model functor Beh, that is,

$$Sem(BMOD) = \{V_s \circ F : Beh(B\Sigma_{IMP}) \to Beh(B\Sigma_{EXP}) \mid F \dashv V_s\}$$

where $BMOD$ is internally correct if there exists a strongly persistent functor

$$F : Beh(B\Sigma_{IMP}) \to Beh(B\Sigma_{BOD})$$

with $F \dashv V_s$ (*cf.* Definition 4.1). Moreover, $(BEQSF, PO_{Beh})$ – a framework whose general practical relevance is discussed after Definition 3.12 – has amalgamations (Theorem 3.15). We are thus assured of the correctness and compositionality of composition, union and actualization of behaviour module specifications as long as we use these operations so that every required pushout is in PO_{Beh} (Theorems 4.5, 4.7 and 4.10). It is even possible to relax this side condition so that a pushout does not need to be in PO_{Beh} whenever its amalgamation property is in fact not needed to prove the respective theorem. An example where this optimization could be applied is the parameter pushout in the union diagram (Figure 15).

Then, with regard to the actualization of module specifications, in Remark 4.9 we pointed out that the last pushout in the construction of the actualized module specification always has amalgamations if the first three do. In the context of $(BEQSF, PO_{Beh})$ we might

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

analogously expect the fourth pushout to be in PO_{Beh} whenever the first three are. But this is not the case, which is shown in the counter example depicted in Figure 17. In this

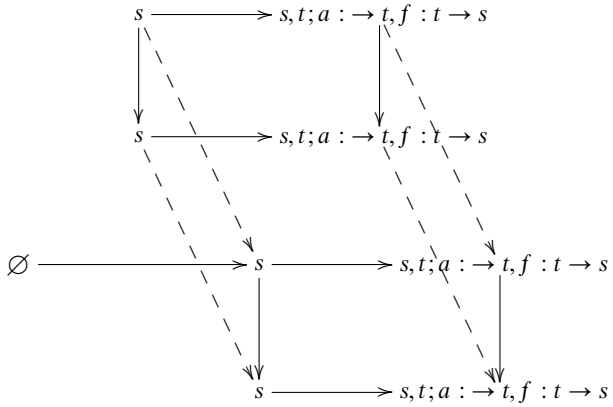


Fig. 17. Counter example behaviour actualization

example the strings represent the specifications, s is an observable and t a non-observable sort, and $a : \to t$, $f : t \to s$ are operation symbols. One can easily see that every square representing the construction squares of the above definition is a pushout in *BSPEC*. Moreover, the first three pushouts trivially satisfy the observation-preserving property because their source specifications only contain observable sorts. But the fourth pushout does not satisfy the condition: the minimal observable $BR\Sigma_{BOD}$-computation $f(a)$ has a non-observable sub-term of sort t the origin of which lies in $B\Sigma_{EXP}$, the source of the pushout. And there is no equivalent clean observation in $BR\Sigma_{BOD}$.

Although the fourth pushout does not satisfy the observation-preserving property, we have amalgamation and extension, as pointed out above and used in the proof of the correctness and compositionality theorem for actualization. Hence, this example shows in an application context that the observation-preserving property is just a sufficient condition for amalgamations and extensions.

We will finish this section with a brief discussion of *VIEWSF*, the behaviour specification frame with syntactic morphisms that only have to preserve the non-observable sorts of a specification.

5.1. *The view specification frame VIEWSF*

$VIEWSF = (View\text{-}BSPEC, Beh)$ (*cf.* Remark 3.3) is defined to consist of the category *View-BSPEC* of behaviour specifications and view-specification morphisms, and the model functor $View\text{-}Beh : View\text{-}BSPEC \to CAT$, which is defined on objects just like *Beh*.

The first difficulty is the *full* definition of *View-Beh*: that is, if $f : BSPEC_1 \to BSPEC_2$ is a view-specification morphism, what is the image of a $View\text{-}Beh(BSPEC_2)$-homomorphism under $View\text{-}Beh(f)$? The problem is that f is defined only on each observable sort. Now, if, for example, we imagine that $BSPEC_1$ and $BSPEC_2$ are one-sorted with a

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

sort symbol of s that is observable in $BSPEC_1$ and non-observable in $BSPEC_2$, then *View-Beh*(f) : *View-Beh*$(BSPEC_2) \rightarrow$ *View-Beh*$(BSPEC_1)$ has to map empty to non-empty homomorphisms, bearing in mind that a homomorphism is a sort-indexed family of mappings between carrier sets. Hence, it is necessary to construct new mappings between carriers that, after forgetting, have become observable.

This problem can be solved by means of a restricted free construction yielding so-called *view functors*.

However, view-functors generally do not have left adjoints. Hence, *VIEWSF* is not liberal in the sense of Definition 2.3. Furthermore, as in *BEQSF*, there are all pushouts in *View-BSPEC*, but they generally do not admit amalgamations or extensions.

To address *this* problem, one could work with an application-oriented variant of the observation-preserving property that mainly requires the horizontal arrows in a diagram like the one in Figure 3 to be behaviour specification morphisms (morphisms in *BEQSF*).

But even these strongly restricted pushouts do not admit amalgamations, and allow extensions only up to isomorphism. (This means that, if there is a strongly persistent left adjoint with respect to f_1 (in the context of Figure 3), we can only guarantee the existence of a *persistent* left adjoint with respect to g_2.) On the other hand, amalgamations and strongly persistent extensions are needed in the correctness and compositionality proofs for composition, union and actualization presented in Section 3. For this reason, we do not present *VIEWSF* and the resulting module theory in this paper. We refer instead to our papers Orejas *et al.* (1989) and Ehrig *et al.* (1991a), where the latter also contains a behavioural version of the modular specification of an airport schedule that was presented in Ehrig and Mahr (1990), using standard equational algebraic specifications.

6. Conclusion

In this paper we have generalized the main parts of the theory of algebraic module specifications, as given in Ehrig and Mahr (1990) for algebraic specifications, to abstract module specifications based on arbitrary specification frames. This is a continuation of the work done for parameterized specifications and parameter passing in Ehrig and Große-Rhode (1994) and Jiménez *et al.* (1995). Moreover, we have presented behaviour specifications in the sense of Nivela and Orejas (1987) and shown counter examples as well as main results under additional assumptions for amalgamation and extension for the corresponding behaviour specification frame.

Of course, the theory of abstract module specifications can also be applied to other specific specification frames including, for example, conditional equational specifications, partial specifications and projection specifications. The corresponding requirements on pushouts, free constructions, amalgamations and extensions have already been verified in Große-Rhode (1989) and Claßen *et al.* (1992). In fact, most of these applications are even simpler because the model functor transforms general colimits on the syntactical level into limits on the semantical level. This allows general amalgamations in the sense of Claßen (1993).

In our paper Ehrig *et al.* (1991a) we have already sketched how to handle abstract specifications with constraints. An elegant treatment of this case requires, however, build-

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

ing a general theory of specification frames extending the theory of indexed categories. Since this goes beyond the scope of this paper we have delayed it for future research. In this paper we have introduced specification frames as indexed categories just as far as is necessary for abstract module specifications.

An important prerequisite for the results of this paper is the internal correctness of abstract module specifications. This property means that the free constructions between the import and the body model categories exist, and that the subclass of strongly persistent functors is non-empty. In some practical applications it seems worthwhile weakening strong persistency not only to persistency, that is, strong persistency up to isomorphism, but also to even weaker notions only requiring consistency and not necessarily completeness. This would also require us to weaken the condition of having amalgamations or extensions on the abstract level. In the case of amalgamation, a possible way to do so is to require the existence of homomorphisms $h_1 : \mathsf{A}_0 \to V_{f_1}(\mathsf{A}_1)$ and $h_2 : \mathsf{A}_0 \to V_{f_2}(\mathsf{A}_2)$ instead of $V_{f_1}(\mathsf{A}_1) = \mathsf{A}_0 = V_{f_2}(\mathsf{A}_2)$. If we consider the pairs (f_1, h_1) and (f_2, h_2) as *generalized morphisms* $(f_1, h_1) : \mathsf{A}_0 \to \mathsf{A}_1$, $(f_2, h_2) : \mathsf{A}_0 \to \mathsf{A}_2$ in the category *GMSF* of generalized morphisms over *SF*, we can regard their pushout as a *generalized amalgamation* of A_1 and A_2 over A_0 via h_1 and h_2. Of course, *SF* has to meet certain conditions for the existence of a natural transformation $\eta : Id \to V_{f_1} \circ F$.

This new concept is called *generalized extension*. It is possible to show that generalized amalgamation implies generalized extension similar to Theorem 2.12 in this paper. These issues are studied in Ehrig *et al.* (1991b). Finally, let us point out that the abstract notion of generalized morphisms originates from a similar notion introduced by Higgins (Higgins 1964). Essentially, this corresponds to the Grothendieck construction leading from indexed categories to fibered categories (Grothendieck 1963).

Appendix

As promised after the definition of free constructions (Definition 2.3), we will prove the following lemma, which is needed in the proof of Theorem 2.12.

Lemma A.1. Let a specification frame $(ASPEC, Catmod)$, an $ASPEC$–morphism $f : A\Sigma_1 \to A\Sigma_2$ and a functor $F : Catmod(AS_1) \to Catmod(AS_2)$ be given such that F is left-adjoint to V_f, the forgetful functor with respect to f. Then $V_f \circ F = Id_{Catmod(AS_1)}$ if and only if the natural transformation $id_{(\_)} : Id_{Catmod(AS_1)} \to V_f \circ F$ with $id_{(\_)}(\mathsf{A}_1) = id_{\mathsf{A}_1}$ for every $\mathsf{A}_1 \in Catmod(A\Sigma_1)$ is a unit of the adjunction.

Proof. The 'if'-direction is trivial, so assume $F \dashv V_f$ in combination with $V_f \circ F = Id_{Catmod(AS_1)}$. We have to show that for all $\mathsf{A}_i \in Catmod(A\Sigma_i)$, $i = 1, 2$, and every $h : \mathsf{A}_1 \to V_f(\mathsf{A}_2)$, there exists a unique $h^\#$ such that $h = V_f(h^\#) \circ id_{\mathsf{A}_1}$, that is, $h = V_f(h^\#)$. Since we already know that there is *a* unit $\eta : Id_{Catmod(A\Sigma_1)} \to V_f \circ F$ with a unique h^* such that $h = V_f(h^*) \circ \eta(\mathsf{A}_1)$, we may stipulate $h^\# =_{def} h^* \circ F(\eta(\mathsf{A}_1))$. Then

$$\begin{aligned} h &= V_f(h^*) \circ \eta(\mathsf{A}_1) \\ &= V_f(h^*) \circ (V_f \circ F)(\eta(\mathsf{A}_1)) \\ &= V_f(h^* \circ F(\eta(\mathsf{A}_1))) \\ &= V_f(h^\#), \end{aligned}$$

https://doi.org/10.1017/S0960129598002606 Published online by Cambridge University Press

so it only remains to show that $h^{\#}$ is unique with respect to this property. To this end, assume $h = V_f(h')$ for some $h' : F(\mathsf{A}_1) \to \mathsf{A}_2$, and let $id^*_{\mathsf{A}_1}$ be unique with respect to $id_{\mathsf{A}_1} = V_f(id^*_{\mathsf{A}_1}) \circ \eta(\mathsf{A}_1)$. We have

$$\begin{aligned} h &= V_f(h') \circ V_f(id^*_{\mathsf{A}_1}) \circ \eta(\mathsf{A}_1) \\ &= V_f(h') \circ (V_f \circ F)(V_f(id^*_{\mathsf{A}_1})) \circ \eta(\mathsf{A}_1) \\ &= V_f(h' \circ (F \circ V_f)(id^*_{\mathsf{A}_1})) \circ \eta(\mathsf{A}_1), \end{aligned}$$

so $h' \circ (F \circ V_f)(id^*_{\mathsf{A}_1}) = h^*$ because of the uniqueness property of h^*. Hence,

$$\begin{aligned} h' &= h' \circ F(id_{\mathsf{A}_1}) \\ &= h' \circ F(V_f(id^*_{\mathsf{A}_1}) \circ \eta(\mathsf{A}_1)) \\ &= h' \circ (F \circ V_f)(id^*_{\mathsf{A}_1}) \circ F(\eta(\mathsf{A}_1)) \\ &= h^* \circ F(\eta(\mathsf{A}_1)) \\ &= h^{\#}. \end{aligned}$$

The proof is thus completed. □

Acknowledgments

We are most grateful to the anonymous referee who has given us a plethora of most worthwhile comments. In addition, we wish to thank Eric Wagner for his support and patience.

This work has been supported partially by the ESPRIT long term research working group 6112 COMPASS and the Spanish CICYT project COSMOS (ref. TIC95-1016-C02-01).

References

Baldamus, M. (1990) Constraints and their Normal Forms in the Framework of Specification Logics (in German). Studienarbeit (TU Berlin).

Benabou, J. (1985) Fibred categories and the foundations of naive category theory. *Journal of Symbolic Logic* **50** 10–37.

Blum, E. K., Ehrig, H. and Parisi-Presicce, F (1987) *JCSS* **34** (2/3) 293–339.

Blum, E. K. and Parisi-Presicce, F (1985) In: Proc. TAPSOFT, Vol. 1. *Springer-Verlag Lecture Notes in Computer Science* **185** 359–373.

Bidoit, M., Hennicker, R. and Wirsing, M. (1994) Characterizing behavioural semantics and abstractor semantics. In: Proc. ESOP'94 (Edinburgh). *Springer-Verlag Lecture Notes in Computer Science* **788** 105–119.

Claßen, I. (1993) *Compositionality of Application Oriented Structuring Mechanisms for Algebraic Specification Languages with Initial Semantics*, Ph. D. thesis, Technische Universität Berlin.

Claßen, I., Große-Rhode, M. and Wolter, U. (1992) Categorical concepts for parameterized partial specifications. Technical Report 92-42, Technische Universität Berlin.

Cornelius, F. (1990a) Behaviour-Ansätze für algebraische Modul-Spezifikationen (in German), Master's thesis, TU Berlin.

Cornelius, F. (1990b) Fallstudie für den Behaviour-Ansatz algebraischer Spezifikationen (in German). Studienarbeit (TU Berlin).

Diaconescu, R., Goguen, J. A. and Stefaneas, P. (1991) Logical support for modularization. Technical report, Oxford University.

Ehrig, H. (1989) Algebraic specification of modules and modular software systems within the framework of specification logics. Technical Report 89-17, TU Berlin.

Ehrig, H., Baldamus, M., Cornelius, F. and Orejas, F. (1991a) Theory of algebraic module specifications including behavioural semantics and constraints. In: Nivat, M., Rattray, C., Rus, T. and Scollo, G. (eds.) *Springer Workshops in Computing* **23** 145–172.

Ehrig, H., Baldamus, M. and Orejas, F. (1991b) New concepts for amalgamation and extension in the framework of specification logics. Technical Report 91-05, TU Berlin.

Ehrig, H. and Große-Rhode, M. (1994) Functorial theory of parameterized specifications in a general specification framework. *Theoretical Computer Science* **135** 221–266.

Ehrig, H., Kreowski, H.J., Thatcher, J.W., Wagner, E.G. and Wright, J.B. (1981) Parameter passing in algebraic specification languages. In: Workshop on Program Specification, Aarhus. *Springer-Verlag Lecture Notes in Computer Science* **134** 322–369. (Also appeared in *Theoretical Computer Science* (1984) **28** 45–81.)

Ehrig, H. and Mahr, B. (1985) *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, EATCS Monographs on Theoretical Computer Science **6**, Springer, Berlin.

Ehrig, H. and Mahr, B. (1990) *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, EATCS Monographs on Theoretical Computer Science **21**, Springer, Berlin.

Ehrig, H. *et al.* (1990) Combining data type and recursive process specifications using projection algebras. *Theoretical Computer Science* **71** 347–380.

Ehrig, H., Pepper, P. and Orejas, F. (1989) On recent trends in algebraic specification. In: Proc. ICALP '89. *Springer-Verlag Lecture Notes in Computer Science* **372** 263–289.

Ehrig, H. and Weber, H. (1985) Algebraic specification of modules. In: Proc. IFIP Work Conf. 85: The Role of Abstract Models in Programming, Wien. (Also as Techn. Report No. 190, FB Informatik, Univ. Dortmund, 1985.)

Ehrig, H. and Weber, H. (1986) Programming in the large with algebraic module specifications. *Information Processing* **86** 675–684. (Invited paper, IFIP'86 World Congress.)

Giarratana, V., Gimona, F. and Montanari, U. (1976) Observability concepts in abstract data type specifications. In: *Proc. 5th Symp. Math. Found. of Comp. Sci.* 576–587.

Goguen, J.A. (1989) A categorical manifesto. Technical Monograph PRG–72, Oxford University Computing Laboratory.

Goguen, J.A. and Burstall, R.M. (1984) Introducing institutions. Proc. Logics of Programming Workshop Carnegie-Mellon. *Springer-Verlag Lecture Notes in Computer Science* **164** 221–256.

Goguen, J.A. and Burstall, R.M. (1992) Institutions: Abstract Model Theory for Specification and Programming. *Journals of the ACM* **39** (1) 95–146.

Goguen, J.A. and Meseguer, J. (1982) Universal realization, persistent interconnection and implementation of abstract modules. In: Proc. IXth ICALP. *Springer-Verlag Lecture Notes in Computer Science* **140** 265–281.

Goguen, J.A., Thatcher, J.W. and Wagner, E.G. (1978) An initial algebra approach to the specification, correctness and implementation of abstract data types. In: Yeh, R. (ed.) *Current Trends in Programming Methodology IV: Data Structuring*, Prentice Hall 80–144.

Gray, J.W. (1965) Fibred and cofibred categories. In: *Proc. Conf. on Categorical Algebra*, Springer 21–83.

Grothendieck, A. (1963) Catégories fibrées et descente. Revetements étales et groupe fondamentale, Séminaire de Géométrie Algébraique du Bois-Marie 1960/61, Exposé VI, Institut des Hautes Études Scientifiques, Paris. (Reprinted in *Springer-Verlag Lecture Notes in Mathematics* (1971) **224** 145-194.

Große-Rhode, M. (1989) Parameterized data type and process specifications using projection algebras. In: Categorical Methods in Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **393** 185–197.

Guttag, J. V. and Horning, J. J. (1978) The algebraic specification of abstract data types. *Acta Informatica* **10** 27–52.

Hennicker, R. and Wirsing, M. (1985) Observational specification: a Birkhoff-theorem. In: Kreowski, H.J. (ed.) *Recent Trends in Data Type Specification; 3rd Workshop on Theory and Appl. of Abstract Data Types, Selected Papers*, Springer Informatik Fachberichte **116** 119–135.

Higgins, P.J. (1963/64) Algebras with a scheme of operators. *Mathematische Nachrichten* **27** 115–132.

Jiménez, R., Orejas, F. and Ehrig, H. (1995) Compositionality and compatibility of parameterization and parameter passing in specification languages. *Math. Struct. in Comp. Science* **5** 283–314.

Johnstone, P. T. and Paré, R. (1978) Indexed categories and their applications. *Springer-Verlag Lecture Notes in Mathematics* **661**.

Löwe, M., Ehrig, H., Fey, W. and Jacobs, D. (1991) On the relationship between algebraic module specifications and program modules. In: Proc. TAPSOFT. *Springer-Verlag Lecture Notes in Computer Science* **494** 83–98.

Mahr, B. (1989) Empty carriers: the categorical burden on logic. In: Ehrig, H., Herrlich, H., Kreowski, H. J. and Preuß, G. (eds.) Categorical Methods in Computer Science – with Aspects from Topology. *Springer-Verlag Lecture Notes in Computer Science* **393** 50–65.

Meseguer, J. (1989) General logics. In: Ebbinghaus, H.-D. *et al.* (ed.) *Logic colloquium '87*, Elsevier Science Publishers B. V., North Holland, 275–329.

Meseguer, J. and Goguen, J. A. (1985) Initiality, induction, and computability. In: Nivat, M. and Reynolds, J. (eds.) *Algebraic Methods in Semantics*, Chapter 14, Cambridge University Press 459–541.

Nivela, P. (1987) *Semántica de Comportamiento en Lenguajes de Especificatión*, Ph. D. thesis, Universitat Politècnica de Catalunya.

Nivela, P. and Orejas, F. (1987) Behavioural semantics for algebraic specification languages. In: Proc. ADT-Workshop, Gullane. *Springer-Verlag Lecture Notes in Computer Science* **332** 184–207.

Orejas, F., Nivela, P. and Ehrig, H. (1989) Semantical constructions for categories of behavioural specifications. In: Categorical Methods in Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **393** 185–197.

Reichel, H. (1981) Behavioural equivalence – a unifying concept for initial and final specification methods. In: *Proc. 3rd Hungarian Comp. Sci. Conf.* 27–39.

Reichel, H. (1984a) Behavioural validity of equations in abstract data types. In: *Contributions to General Algebra 3, Proc. of the Vienna Conf.*, Teubner 301–324.

Reichel, H. (1984b) *Structural Induction on Partial Algebras*, Mathematical Research – Mathematische Forschung **18**, Akademie-Verlag, Berlin.

Rus, T. (1979) *Data Structures and Operating Systems*, John Wiley & Sons.

Rus, T. (1990) Steps towards algebraic construction of compilers. Technical report, Univ. of Iowa.

Sannella, D. T. and Tarlecki, A. (1984) Building specifications in an arbitrary institution. In: Proc. Int. Symposium on Semantics of Data Types. *Springer-Verlag Lecture Notes in Computer Science* **173** 337–356.

Sannella, D. T. and Tarlecki, A. (1985) Some thoughts on algebraic specification. In: *Proc. 3rd Workshop on Theory and Application of Abstract Data Types, Bremen*, Springer, Informatik Fachberichte **116** 31–38.

Sannella, D. T. and Tarlecki, A. (1987a) On observational equivalence and algebraic specification. *JCSS* **34** 150–187.

Sannella, D. T. and Tarlecki, A. (1987b) Towards formal development of programs from algebraic specifications: implementations revisited. In: Proc. Joint Conf. on Theory and Practice of Software Development, Pisa (Extended Abstract). *Springer-Verlag Lecture Notes in Computer Science* **249** 96–110. (Full Version in *Acta Informatica* (1988) **25** 233–281.)

Sannella, D. T. and Wirsing, M. (1983) A kernel language for algebraic specification and implementation. In: Proc. 1983 Int. Conf. on Foundations of Computation Theory. *Springer-Verlag Lecture Notes in Computer Science* **158** 413–427.

Tarlecki, A., Burstall, R. M. and Goguen, J. A. (1991) Some fundamental algebraic tools for the semantics of computation. Part III: Indexed categories. *Theoretical Computer Science* **91** 239–264. (Also appeared as technical report ECS-LFCS-89-90, Univ. of Edinburgh, 1989.)

Thatcher, J. W., Wagner, E. G. and Wright, J. B. (1978) Data type specification: Parameterization and the power of specification techniques. In: 10th Symp. Theory of Computing 119–132. (ACM Trans. Prog. Languages and Systems 4 (1982) 711–732.)

Wand, M. (1979) Final algebra semantics and data type extensions. *JCSS* **19**.

Weber, H. and Ehrig, H. (1986) Specification of modular systems. *IEEE Transactions on Software Engineering* SE-12(7) 784–798.

Zilles, S. N. (1974) Algebraic specification of data types. Technical Report 11, MIT Project MAP Progress Report 28–52.