Behavioural approach to modular system specification

H. Ehrig
M.P. Nivela
F. Orejas

Report LSI–88–24

Abstract: The algebraic approach to modular specification,as developed by the group of Berlin, is joined with the behavioural approach to algebraic specifications developed in Barcelona. In the joint approach module specifications are provided with a behavioural functorial semantics. As main results it is shown that basic interconnection mechanisms, like composition, union and actualization, are correctness preserving and compositional w.r.t. the behavioural semantics.

In practice also the case of view actualization is of great interest where non observable actual sorts can be bound to observable formal sorts. Also in this case, which requires behavioural view functors instead of forgetful functors, it is discused how to extend the basic constructions and results.

Resum: L'enfoc algebraic de les especificacions modulars desenvolupades a Berlin es combina amb l'enfoc de comportament desenvolupat a Barcelona. En aquest enfoc conjunt les especificacions algebraiques són dotades de semántica de comportament. Els resultats principals mostren que els mecanismes básics de conexió, com la composici'ó, l'unió i la instanciació, preserven la correctesa i la composicionalitat respecte la semántica de comportament.

A la práctica és molt interesant el cas de la instanciació "vista", en el que sorts observable en el parámetre formal poden ser associats a sorts no observables del parmetre real. En aquest cas es necessari utilitzar funtors "vista" en lloc de funtors oblit i s'estudia com extendre els resultats básics.

Behavioural Approach to Modular System Specification

H. Ehrig *, Mª P. Nivela, F. Orejas
Universitat Politècnica de Catalunya
Barcelona, Spain
July, 1.988

Abstract

The algebraic approach to modular specifications as developed by the first author during the last five years is joined with the behavioural approach to algebraic specifications introduced recently by the last two authors. In the joint approach module specifications are provided with a behavioural functorial semantics. As main results we show that basic interconnection mechanisms, like composition, union and actualization are correctness preserving and compositional w.r.t the behavioural semantics. In practice also the case of view actualization is of great interest where non observable actual sorts can be bound to observable formal sorts. Also in this case, which requires behavioural view functors instead of forgetful functors, we discuss how to extend the basic constructions and results. Most constructions and results are illustrated by a behavioural module specification of a modular airport schedule system

.. Introduction

Module specifications were defined by Blum, Ehrig and Parisi-Presicce [BEP 87] as a concept for upporting the structured design of large specifications. In this framework, a module may be seen as a arameterized abstract data type extended by an import and an export interface. The export interface is the isible part describing what the module offers to a user. This allows to interconnect modules in a flexible ıanner and to build up modular software systems in stepwise style.

---

on visit, permanent address Technical University of Berlin

2. Behavioural semantics of module specification

In this section we will define the basic concepts concerning behavioural semantics of modules. In the first subsection we present the basic definitions and results about behavioural semantics: categories of behaviours, satisfaction relation associated to a set of equations, behaviour specification morphisms and its associated free and forgetful functors and pushouts results. In the second subsection the notion of behaviour module specification, its semantics and correctness concept is presented. Finally, the third subsection deals with the airport schedule system exemple.

2.1 General notion of behavioural semantics

2.1.1 Behavioural equivalence

Definition 2.1.1.1

A **behaviour signature** BSig is a triple BSig = (Obs,S,Σ) with Obs $\subseteq$ S where Sig = (S, Σ) is a usual signature (see [EM 85]).The sorts in Obs are called **observable sorts.**

To every signature BSig, we may associate two categories of models: Alg(Sig) the well-known category of Sig-algebras and Sig-homomorphisms (see [EM 85]), and the one we will use for defining behavioural semantics, Beh(BSig) as introduced in [NO 87]. In this category models are Sig-algebras, as in Alg(Sig), but morphisms are different. To avoid confusion, from now on, morphisms in Alg(Sig) will be called Sig-homomorphisms, while morphisms in Beh(BSig) will be called Sig-behaviour morphisms.

Definition 2.1.1.2

Let A be a Sig-algebra. A **computation** (resp. an **observable computation**) over A is a term in $T_\Sigma(A_{Obs})$ (resp. $T_\Sigma(A_{Obs})_s$ with $s \in Obs$).

Definition 2.1.1.3

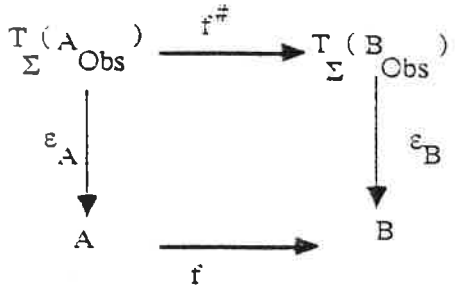
Given two Sig-algebras A and B, a **Sig-behaviour morphism** $f: A \to B$ is a family of functions $f=\{f_s\}_{s\in Obs}$ such that for all observable computations $t \in T_\Sigma(A_{Obs})_s$, $s \in Obs$, the following equality holds

$$f_s(\varepsilon_A(t)) = \varepsilon_B(f^\#{}_s(t))$$

where $f^\#: T_\Sigma(A_{Obs}) \to T_\Sigma(B_{Obs})$ is the unique Sig-homomorphism which extends f and ε_A is the evaluation function,i.e., the unique Sig-homomorphism extending the inclusion of A_{Obs} into A. Sig-algebras together with Sig-behaviour morphisms form the category **Beh(BSig)**.

Remark 2.1.1.4

- It can be noted that if Obs coincides with S then Beh(BSig) is exactly the same as Alg(Sig).
- A Sig-behaviour morphism establishes a relationship between the observable computations and their results in A and B respectively. To every observable computation t over A corresponds the observable computation $f^\#(t)$ over B and to the result $\varepsilon_A(t)$ of this computation over A corresponds the result $\varepsilon_B(f^\#(t))$ in B. Thus an Obs-indexed family $f = \{f_s\}_{s\in S}$ is a Sig-behaviour morphism if these observable computations over A yield in B the same value as in A, up to the transformation determined by f. This means that the following diagram commutes:



for all sorts in Obs.

If the converse holds, that is, if all the observable computations over B yield in A the same value as in B up to the transformation determined by f, and f itself is a bijection, then A and B give the same answers to the same questions, that is, they *show the same observable behaviour*. Hence behavioural equivalence is characterized by isomorphism in the category Beh(BSig). In particular, isomorphism in Beh(BSig) coincides with the notion of behavioural equivalence from [MG 85, HWi85, SWi83, ST 85].

• A Sig-behaviour morphism f such that f_s is bijective for every s in Obs is Sig-behaviour isomorphism in the category Beh (BSig).

Definition 2.1.1.5

A and B are **behaviourally equivalent**, denoted $A \equiv_{BSig} B$, if there exists a Sig-behaviour isomorphism $f : A \to B$.

Definition 2.1.1.6

Let BSig = (Obs,S, Σ) be a behaviour signature, X_{Obs} an Obs-indexed family of variables and z a variable symbol of sort s not belonging to X. A **Sig-context over the sort** s (resp. **Sig-context** for a Sig-algebra A **over the sort** s) is a term $c[z] \in T_\Sigma(X_{Obs} \cup \{z\})_{s'}$ with $s' \in$ Obs (resp. $c_A[z] \in T_\Sigma(A_{Obs} \cup \{z\})_{s'}$ with $s' \in$ Obs).

Given a context c[z], (resp. $c_A[z]$) and a term t in $T_\Sigma(X_{Obs})_s$ (resp. in $T_\Sigma(A_{Obs})_s$), by c[t] (resp. $c_A[t]$) we denote the **application of the context over** t, that is, $\bar{\sigma}(c[z])$ (resp. $\bar{\sigma}(c_A[z])$), where σ is the assigment $\sigma : X_{Obs} \to T_\Sigma(X_{Obs})$ (resp. $\sigma : \{z\} \to T_\Sigma(X_{Obs})$) defined by σ(z) = t and σ(x) = x for every x in X_{Obs} (resp. σ(z) = t).

Definition 2.1.1.7

Let A be a Sig-algebra and e: λY.t1=t2 a Sig-equation of sort s. A **behaviourally satisfies** e, denoted $A \models_B e$, if A satisfies $\lambda X_{Obs}.c[\bar{\sigma}(t1)]=c[\bar{\sigma}(t2)]$ for every Sig-context c[z] over the sort s and every assignment $\sigma : Y \to T_\Sigma(X_{Obs})$.

Remark 2.1.1.8

• If we consider contexts as observations, it would seem quite natural to assert that "an algebra A behaviourally satisfies the equation e if A satisfies c[t1] = c[t2] for every Sig-context c[y] over the sort s", that is, A satisfies every kind of observable consequence of e. In this way, an equation of observable sort would be satisfied in the classical sense and an equation of non observable sort would be satisfied up to observable contexts. This is the notion of behavioural satisfaction defined in [RE 84]. However, in our framework, this definition is not appropriate because the replacement of the variables in the equation by "non obsevable inputs " (more specifically, by non observable junk) can cause that behavioural equivalent algebras do not behaviourally satisfy the same equations. This problem is not present in [RE 84] because of his different notion of behavioural equivalence. So we will only allow to substitute variables by values finitely generated from the observable ones. This is coherent with the intuitive idea

that "we do not care about the nonobservable part".

• From now on, in order to enhance readability, we will omit the explicit quantification of variables in equations, though technically needed [GM 81].

Definition 2.1.1.9

A **behaviour specification** BSPEC is a 4-tuple, BSPEC = (Obs, S, Σ, E), where BSig =(Obs, S, Σ) is a behaviour signature and E a set of Sig=(S, Σ)-equations. **Beh(BSPEC)** is the full subcategory of Beh(BSig) of all Sig-algebras which behaviourally satisfy the equations in E.

In what follows we will shortly write BSPEC = (Obs, SPEC) with SPEC = (S, Σ, E) and Obs $\subseteq$ S.

2.1.2 Behaviour specification morphisms, parameterized behaviour specifications, free constructions and pushouts

Definition 2.1.2.1

Let BSPECi=(Obsi,SPECi) i = 1, 2 be two behaviour specifications with SPECi = (Si, Σi, Ei). A **behaviour specification morphism** h : BSPEC1 → BSPEC2 is a specification morphism h : SPEC1 → SPEC2 (i.e. h is a signature morphism with E2 |- h(E1)) such that h(Obs1) $\subseteq$ Obs2 and h(S1-Obs1) $\subseteq$ S2-Obs2 (i. e. observable and non observable sorts are preserved).

Remark 2.1.2.2

• To a behaviour specification morphism h : BSPEC1 → BSPEC2 it can be associated a forgetful functor BU_h : Beh(BSPEC2) → Beh(BSPEC1) defined as usual. Due to h(Obs1) $\subseteq$ Obs2 it holds that if A2 and A2' are two algebras in Beh(BSPEC2) such that A2 $\equiv_{BSPEC2}$ A2' then BU_h(A2) $\equiv_{BSPEC2}$ BU_h(A2').

• For the existence of free and forgetful functors is not needed that h(S1-Obs1) $\subseteq$ S2-Obs2 [NO 87].

Associated to a behaviour specification morphism h there is a free functor, $BFree_h$, left adjoint to the forgetful functor BU_h.

Theorem 2.1.2.3

Let BSPECi = (Obsi, SPECi), SPECi = (Si, Σi, Ei), i = 1, 2 be two behaviour specifications and h : BSPEC1 → BSPEC2 a behaviour specification morphism. For every algebra A in Beh(BSPEC1), the algebra defined by

$$BFree_h(A) = T_{\Sigma 2}(A_{Obs1})/\equiv_{eobs(A)\div E2}$$

(interpreting the values $a \in A_s$, $s \in$ Obs1, as values of sort h(s)) is a free construction over A with respect to U_h which extends to a free functor $BFree_h$: Beh(BSPEC1) → Beh(BSPEC2).

Remark 2.1.2.4

• Let us observe that if Obs1 = S1 then for every SPEC1-algebra A, $BFree_h$(A) is the usual free construction.

• Even if h is not preserving non observable sorts, i. e. h(S1 -Obs1) $\not\subset$ S2 - Obs2, the forgetful and

free functors U_h and $BFree_h$ exist [NO 87].

• The behavioural equivalence relation may be extended uniformly from algebras to functors, that is, behavioural equivalence of functors coincides with natural isomorphism. So, if F and F' are two functors from Beh(BSPEC1) to Beh(BSPEC2) we will say that F and F' are behaviourally equivalent if they are naturally isomorphic and we will denote that by $F \equiv F'$. Therefore, we immediately have that any functor behaviourally equivalent to a free functor is also free.

• Moreover, a functor $F : Beh(BSPEC1) \rightarrow Beh(BSPEC2)$ is persistent with respect $U : Beh(BSPEC2) \rightarrow Beh(BSPEC1)$ if $U \cdot F \equiv Id_{Beh(BSPEC1)}$.

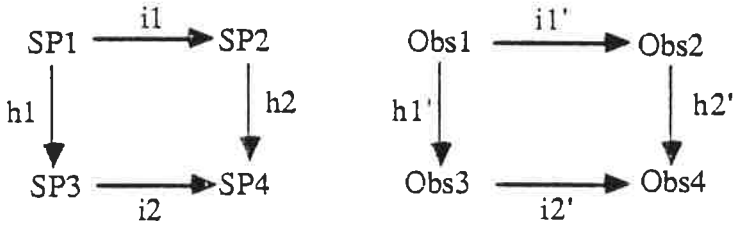
Definition 2.1.2.5

Let BSPECi = (Obsi, SPECi), SPECi = (Si, Σi, Ei), Obsi ⊆ Si, i = 1, 2 be two behaviour specifications. A **parameterized behaviour specification** BPSPEC is a pair of behaviour specifications BPSPEC = (BSPEC1, BSPEC2) such that Obs1 = S1 and SP1 ⊆ SP2. BSPEC1 is called **formal parameter specification** and BSPEC2 the **target specification.** The **behavioural semantics** of BPSPEC is the (class of) free functor(s) associated to the inclusion specification morphism.

Let **Set**, **SPEC** and **BSPEC** be the categories of sets and maps, specifications and specification morphisms and behaviour specifications and behaviour specification morphisms respectively. We show tha we have pushouts in the category **BSPEC**, similar to those in **SPEC**.

Proposition 2.1.2.6

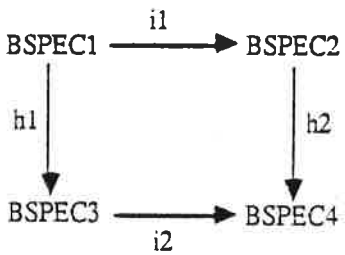
Let BSPECi = (Obsi, SPi), i = 1, 2 be two behaviour specifications and i1 : BSPEC1 → BSPEC2, h1 : BSPEC1 → BSPEC3 two behaviour specification morphisms.The behaviour specification BSPEC4 = (Obs4, SP4), defined by Obs4 being the pushout of i1' and h1' in the category Sets, SP4 the pushout of i1 and h1 in the category **SPEC**, together with the behaviour specification morphisms h2 : BSPEC2 → BSPEC4 and i2 : BSPEC3 → BSPEC4 defined by the correponding pushouts diagrams below,



is a pushout on **BSPEC**, where $i1' = i1_{|Obs1}$, $h1' = h1_{|Obs1}$, $i2' = i2_{|Obs2}$ and $h2' = h2_{|Obs2}$ are the restrictions of i1, h1, i2 and h2 to the their respective observable sorts.

Definition 2.1.2.7

A pushout diagram



satisfies the **observation preserving property** if for every minimal observable computation $t \in T_{\Sigma 4}(X_{Obs4})$, (i.e. no subterm of t, different from a variable, is an observable computation) and

every $s \in S4\text{-}Obs4$ being the sort of a non observable subterm of t it holds that $s \in S4 - i2 \circ h1(S1)$ or, equivalently, $t \in T_{i2(\Sigma 3)} \cup T_{h2(\Sigma 2)}$.

Remarks 2.1.2.3

- Observation preserving will be needed to obtain most of our results. In fact, if a diagram does not satisfy it then there is no Amalgamation Lemma associated to it.
- The problem when this property is not satisfied is that BSPEC3 and BSPEC2 may add more ways of "observing" the values of the non observable sorts of BSPEC1. This makes incompatible the semantics of (BSPEC3, BSPEC4) and (BSPEC1, BSPEC2).
- If Obs1 = S1 then the property trivially holds. Under this assumption all results about parameter passing were obtained in [NO 87].

2.2 General notion of behaviour module specification

The concept of a behaviour module is an extension of the module concept in [BEP 87] in which observable sorts are distinguished in each part in order to use the concepts of behavioural specifications in the sense of [NO 87].

Definition 2.2.1

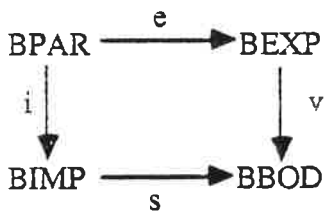
A **behaviour module specification** BMOD is a 8-tuple

$$BMOD = (BPAR, BEXP, BIMP, BBOD, e, s, i, v)$$

where:

a) BPAR = (ObsPAR, PAR) is called **behaviour parameter specification**
BEXP = (ObsEXP, EXP) is called **behaviour export interface specification**
BIMP = (ObsIMP, IMP) is called **behaviour import interface specification**
BBOD = (ObsBOD, BOD) is called **behaviour body specification**

b) e, s, i, v are behaviour specification morphism such that $v \circ e = s \circ i$, that is, the following diagram commutes



We will just write BMOD = (BPAR, BEXP, BIMP, BBOD) if the behaviour specification morphisms e, s, i and v are clear in the context.

Forgetting the distinction of observable sorts we obtain a usual module specification MOD = (PAR, EXP, IMP, BOD, e, s, i, v) in the sense of [BEP 87].

Definition 2.2.2

Let BMOD = (BPAR, BEXP, BIMP, BBOD, e, s, i, v) be a behaviour module specification. The **behavioural semantics** of BMOD is the functor $BSEM = U_V \circ BFree_S$ where $BFree_S$ is the free

functor w.r.t. the forgetful functor U_S (see 2.1.2.3).

Definition 2.2.3

A behaviour module specification BMOD = (BPAR, BEXP, BIMP, BBOD, e, s, i, v) is **behaviourally correct** if $BFree_S$ is strongly persistent, that is, $BU_S \circ BFree_S = Id_{Beh(BIMP)}$.

Proposition 2.2.4

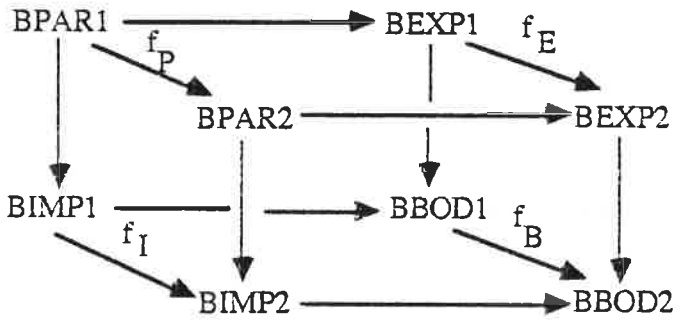
If BMOD is a correct behaviour module specification then $BU_e \circ BSEM = BU_i$.

Proof

Due to that $v \circ e = s \circ i$ we have $BU_e \circ BSEM = BU_e \circ BU_v \circ BFree_S = BU_i \circ BU_S \circ BFree_S$. Moreover, by persistency of $BFree_S$, we have $BU_e \circ BSEM = BU_i$.

Definition 2.2.5

Given two behaviour module specifications BMODj = (BPARj, BEXPj, BIMPj, BBODj), j = 1,2, a **behaviour module specification morphism** f : BMOD1 → BMOD2 is a 4-tuple f = (f_P, f_E, f_I, f_B) of behaviour specification morphisms such that the following diagram commutes



Definition 2.2.6

A behaviour module specification morphism f : BMOD1 → BMOD2 is **coherent** if the semantics BSEM1 and BSEM2 of BMOD1 and BMOD2 are compatible with the corresponding forgetful functors up to behavioural equivalence, i. e.

$$BSEM1 \circ BU_{f_I} \equiv BU_{f_E} \circ BSEM2$$

2.3 Example

The airport schedule system specified in [EW 86] can be studied under the behavioural approach establishing the following observable sorts in each part of the module.

aps-bmod = (aps-par, aps-exp, aps-imp, aps-bod)

aps-par = **Bool** +
bspec
sorts obs flight# {*flight number*}, dest {*destination*}, dep {*departure*},
plane# {*plane number*}, type, seats {*number of seats*}
end bspec

```
aps-exp = aps-par +
            bspec
              sorts  non  obs  aps {airport schedule}
              ops
                create : → aps
                schedule : flight# dest dep plane# type seats aps → aps
                search : flight# aps→ bool                    cancel : flight# aps→ aps
                return_dest : flight# aps→ dest               return_dep : flight# aps→ dep
            end bspec


aps-imp = aps-par +
            bspec
              sorts  obs   fs {flight schedule}, ps {plane schedule}

              ops    create_fs :→ fs                  create_ps :→ ps
                     add_fs : flight# dest dep fs → fs  add_ps : plane# type seats ps → ps
                     search_fs : flight# fs → bool      search_ps : plane# ps → bool
            end bspec


aps-bod = aps-exp + aps-imp +
            bspec
              ops    tup : fs ps → ps
              ecns   fn : flight#; dt : dest; dp : dep; pn : plane#; t : type; s : seats; xfs : fs; xps : ps
                     create = tup(create_fs, create_ps)
                     schedule(fn, dt, dp, pn, t, s, tup(xfs, xps)) =
                          = if search_fs(fn, xfs) or search_ps(pn, xps) then tup(xfs, xps)
                            else tup(add_fs(fn, dt, dp, xfs), add_ps(pn, t, s, xps))
                     .....
            end bspec
```

Airport schedules are constructed by means of a hidden function "tup" of the aps-bod part which builts airport schedules as tuples of flight schedules (fs) and plane schedules (ps). This function is hidden because in general it constructs inconsistent elements.

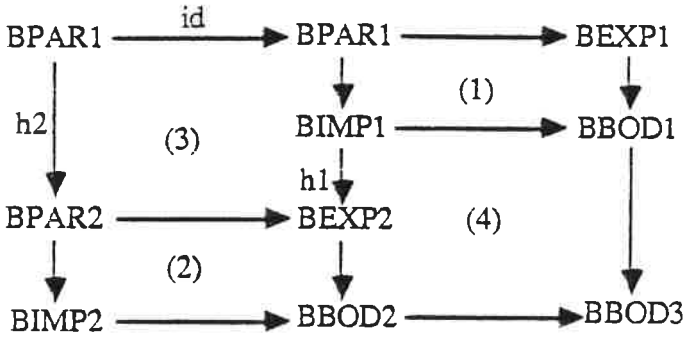
The sorts "flight#", "dest", "dep", "plane#", "type", "seats", "fs" and "ps" are considered as observable and the sort "aps" as non observable. The behavioural semantics of the **aps-bmod** is the functor BSEM which transforms aps-imp-algebras into aps-exp-algebras. Given an aps-imp-algebra A it constructs the aps-export-algebra U ∘ Free(A). Due to that the sort "aps" is non observable it is not necessary, as in the usual semantics of modules [BEP 87], to remove the junk (i.e. inconsistent airport schedules) that can be generated in the body, because the two algebras U ∘ Free(A) and Restr ∘ U ∘ Free(A) are behaviourally equivalent, where Restr is the restriction functor of [BEP 87].

3. Composition, union and actualization

In this section we introduce as basic interconnection mechanisms for behavioural module specifications the operations composition, union and actualization. They are extension of the corresponding operations for usual module specifications as introduced in [BEP 87]. We are able to show that also the main results concerning correctness and compositionality of these operations can be extended using a new Amalgamation and Extension Lemma for the behavioural case with behaviour specification morphisms, provided that the corrsponding constructions are behavioural consistent.

Definition 3.1

The **composition** of two behaviour module specifications BMODi = (BPARi, BEXPi, BIMPi, BBODi), i = 1,2, via a pair h = (h1, h2) of behaviour specification morphisms making diagram (3) below commutative, is the behaviour module specification BMOD3 = (BPAR3, BEXP3, BIMP3, BBOD3) given by the outer square in



with BPAR3= BPAR1, BEXP3 = BEXP1, BIMP3 = BIMP2 and BBOD3 the pushout object in (4), written BMOD3 = BMOD2 $\circ_h$ BMOD1.

The composition is called **behavioural consistent** if pushout (4) satisfies the observation preserving property (see 2.2.2.7).

Theorem 3.2 (Correctness and compositionality of composition)

Given correct behaviour module specifications BMOD1 and BMOD2 and a pair h = (h1, h2) such that the composition BMOD3 = BMOD2 $\circ_h$ BMOD1 is defined and behavioural consistent, we have

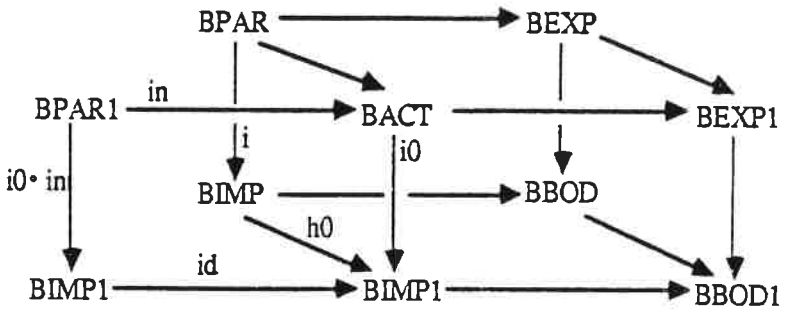
1. **Correctness** BMOD3 is correct
2. **Compositionality** BSEM3 = BSEM1 $\cdot$ BU_{h1} $\cdot$ BSEM2
   where BSEMj, j = 1, 2, 3 is the behavioural semantics of BMODj

Proof idea

Extend the proof of theorem 3.3 in [BEP 87] to the behavioural case using the Behavioural Extension Lemma (see Appendix) applied to pushout (4) in definition 3.1 satisfying the observation preserving property (see 2.2.2.7).

Definition 3.3

The **actualization** of a behaviour module specification BMOD = (BPAR, BEXP, BIMP, BBOD) by a parameterized behaviour specification BPSPEC1 = (BPAR1, BACT) via a behaviour specification morphism h : BPAR → BACT is the behaviour module specification BMOD1 = (BPAR1, BEXP1, BIMP1, BBOD1) given by the outer front square in



where BEXP1, BIMP1 and BBOD1 are constructed as pushouts in the top, left and bottom part respectively, written BMOD1 = $BMOD_h$(BPSPEC1).

The actualization is called **behavioural consistent** if the bottom pushout satisffies the observation preserving property (see 2.2.2.7).

Theorem 3.4 (Correctness and compositionality of actualization)

Given a correct behaviour module specification BMOD, a parameterized behaviour specification BPSPEC1 and a morphism h such that the actualization BMOD1 = $BMOD_h$(BPSPEC1) is defined and behavioural consistent, we have

1. **Correctness** BMOD1 is correct
2. **Compositionality** BSEM1(I $\oplus_P$ F(P1)) = BSEM(I) $\oplus_P$ F(P1)
   where BSEM, BSEM1 and F are the behavioural semantics of BMOD, BMOD1 and BPSPEC1 respectively. The operation $\oplus$ denotes the behavioural amalgamated sum of a BIMP-algebra I, BPAR-algebra P and BACT-algebra F(P1) on the left hand side defining a BIMP1-algebra I $\oplus_P$ F(P1), and similarly a BEXP1-algebra on the right hand side.

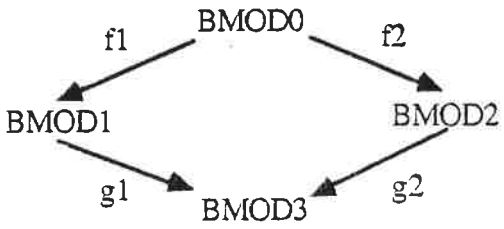
Proof idea

Extend the corresponding proof in [PP 87] for the non behavioural case using the Behavioural Extension and Amalgamation Lemma (see Appendix) where the pushouts in diagram 3.3 satisfy the observation preserving property by behavioural consistency of actualization.

Definition 3.5

Given behaviour module specifications BMODj, j= 0, 1, 2, and two behaviour module specification morphisms fj : BMOD0 → BMODj, j = 1, 2, the **union** of MOD1 and MOD2 via MOD0 and (f1, f2) is given by the pushout BMOD3, written BMOD3 = BMOD1 $+_{BMOD0}$ BMOD2, in the category of behaviour module specifications and behaviour module specification morphisms.

The union is called **behavioural consistent** if the pushout satisfies the observation preserving property in all componenets (see 2.1.2.7)



Remark 3.6

• Similar to [PP 87] actualization can be considered as a special case of union also for behaviour module specifications.

Theorem 3.7 (Correctness and compositionality of union)

Given correct behaviour module specifications BMODj, j = 0, 1, 2, and coherent behaviour module specification morphisms fj, such that the union BOMD3 = BMOD1 $+_{BMOD0}$ BMOD2 is behavioural consistent, then we have

1. Correctness BMOD3 is correct
2. Compositionality BSEM3 = BSEM1 $\oplus_{BMSEM0}$ BSEM2
   where BSEMj is the behavioural semantics of BMODj for j = 0, 1, 2, 3 and $\oplus$ the behavioural amalgamated sum of functors

Proof idea

Extend the proof of theorem 4.12 in [BEP 87] to the behavioural case using the Behavioural Extension Lemma (see Appendix) where all the pushouts satisfy the observation preserving property.

Example 3.8 (Interconnection of behaviour modules in the airport schedule system)

The example in section 2.3 can be enriched considering the following behaviour modules **bool-bmod, fs-bmod, ps-bmod:**

```
bool-bmod = (bool-par, bool-exp, bool-imp, bool-bod)
bool-par = (Ø, Ø, Ø, Ø)
bool-exp = bspec
             sorts obs bool
             ops   true, false : → bool
                   not : → bool
                   and, or : bool bool → bool
             ecns  {the usual ones}
           end bspec

bool-imp = bool-par
bool-bod = bool-exp
```

```
fs-bmod = (fs-par, fs-exp, fs-imp, fs-bod)
fs-par = bool-exp +
        bspec
            sorts obs flight#, dest, dep
            ops   eq : flight# flight# → bool
            ecns  fn : flight#
                  eq(fn, fn) = true
        end bspec
fs-exp = fs-par +
        bspec
            sorts obs fs {flight schedule}
            ops   create_fs : → fs
                  add_fs : flight# dest dep fs → fs
                  search_fs : flight# fs→ bool
            ecns  .....
        end bspec

fs-imp = fs-par
fs-bod = fs-exp +
        bspec
            ops   tab : flight# dest dep fs → fs
            ecns  .....
        end bspec
```

```
ps-bmod = (ps-par, ps-exp, ps-imp, ps-bod)
ps-par = bool-exp +
        bspec
            sorts obs plane#, type, seats
            ops   eq : plane# plane# → bool
            ecns  pn : plane#
                  eq(pn, pn) = true
        end bspec
ps-exp = ps-par +
        bspec
            sorts obs ps {plane schedule}
            ops   create_ps : → ps
                  add_ps : plane# type seats ps → ps
                  search_ps : plane# ps→ bool
            ecns  .....
        end bspec

        ps-imp = ps-par
ps-bod = ps-exp +
        bspec
            ops   tab : plane# type seats ps → ps
            ecns  .....
        end bspec
```

The inclusions i1 : **bool-bmod → fs-bmod** and i2 : **bool-bmod → ps-bmod** are coherent behaviour specification morphisms such that the module **fsps-bmod** defined as the union

$$\textbf{fsps-bmod} = \textbf{fs-bmod} +_{\textbf{bool-bmod}} \textbf{ps-bmod}$$

is well defined, behaviouiral consistent and correct (see thm. 3.6). Now we are able to define the composition

$$\textbf{aps-system-bmod} = \textbf{aps-bmod} \circ_h \textbf{fsps-bmod}$$

where h = (id, id) is a pair of identity morphisms. The **aps-system-bmod** module is behavioural coherent and by theorem 3.2 is again correct and its behavioural semantics can be composed by those of

the components. Behavioural consistency is satisfied by all this modules due to part 3 of remark 2.1.2.8.

4. Composition and actualization with view morphisms

The operations of composition and actualization, as defined in the previous section are sometimes insufficient to work with module specifications with true behavioural semantics. For instance, the fact that the fitting morphism used for actualization would be a behaviour specification morphism, together with the fact that all parameter sorts are observable, would impose, to satisfy the behavioural consistency property of actualization, that all sorts of the actual parameter bounded to sorts of the module parameter should be observable, since behaviour specification morphism preserve observable sorts.

This, for example, would mean that it would not be possible to instantiate the **aps-system-bmod** module with a flight number which is a sequence of characters, if the sort sequence is not observable.

A similar situation occurs when trying to compose two modules. If for semantic reasons we have defined an import sort of the first module to be observable, the corresponding output sort of the second module would need to be also observable, if we want to satisfy the behavioural consistency property for composition. For instance, if in the **aps-bmod** module we have declared the sort fs to be observable, since otherwise the semantics of import algebras would be too poor because there are not enough observations on this sort, then it would be impossible to compose **aps-bmod** with **fs-bmod** if in the latter fs has been declared non observable.

To overcome this problem, it is sensible to allow working with a more general notion of behaviour specification morphism, namely what we have called *view specification morphisms* .

Definition 4.1

Let BSPECi=(Obsi,SPECi) i = 1, 2 be two behaviour specifications with SPECi = (Si, Σi, Ei). A **view specification morphism** h : BSPEC1 → BSPEC2 is a specification morphism h : SPEC1 → SPEC2 such that h(S1-Obs1) ⊆ S2-Obs2.

Remark 4.2

• A view specification morphism not necessarily satisfies h(Obs1) ⊆ Obs2 as required for behaviour specification morphisms.

• Let us observe that if h is a view specification morphism then it has no sense to consider U_h because in Obs2 there can be "less" observable sorts than in Obs1. This means that when "forgetting" a BSPEC2-behaviour morphism $f = \{f_s : A2_s \to A2'_s\}_{s \in Obs2}$ there can exist some sort $s \in Obs1$ such that $h(s) \notin Obs2$ and , therefore, $f_{h(s)}$ is not defined. Passing from BSPEC2 to BSPEC1 behaviours can be done by a functor $View_h$, called view functor, which builts up a BSPEC1-behaviour morphism from a BSPEC2-behaviour morphism and describes how the models of Beh(BSPEC2) are "seen" from the BSPEC1 "point of view". First of all a special realization of the behaviour of an algebra A2 in Beh(SPEC2) is constructed, that is, an algebra behaviourally equivalent to A2 but in the category Alg(SPEC2* + h(E1)) defined below, in such a way that BSPEC2-behaviour morphisms can be extended to usual SPEC2* + h(E1) homomorphisms. Secondly, once we are in Alg(SPEC2* + h(E1)), a forgetful functor from Alg(SPEC2*+h(E1)) to Beh(BSPEC1) is applied.

• Pushouts for this kind of morphisms would be defined in a similar way as for behaviour specification morphisms. However, in this section we will only work with "mixed" diagrams , i.e. diagrams in which two morphisms are behaviour specification and the other two are view specification

morphisms. Formally, this poses no problem since the formers are a special case of the latters. Also, the observation preserving property can be defined acordingly.

Definition 4.3

Let BSPEC = (Obs, SPEC) be a behaviour specification with SPEC = (S, Σ, E). The specification **behaviourally derived from** BSPEC is SPEC* = (S, Σ, E*) where

$$E^* = \{t_1 = t_2 \mid t_1, t_2 \in T_\Sigma(X_{Obs})_s, s \in Obs, E \vdash t_1 = t_2\}$$

is the set of all equations of observable sort over observable variables which may be deduced from E.

Remark 4.4

• The category Beh(BSPEC) has the same objects as Alg(SPEC*) but behaviour morphisms are in a sense "weaker" than homomorphisms: every Σ-homomorphism may be seen as a behaviour morphism forgetting about the functions defined in non observable sorts. That allows to consider a forgetful functor

$$U_{BSPEC} : Alg(SPEC^*) \rightarrow Beh(BSPEC)$$

defined by $U_{BSPEC}(A) = A$ for every A in Alg(SPEC*) and to every Σ-homomorphism f corresponds $U_{BSPEC}(f) = f|_{Obs}$, that is the same f viewed as a Σ-behaviour morphism by its restriction to the observable sorts.

Definition 4.5

Let BSPEC = (Obs, SPEC) be a behaviour specification with SPEC = (S, Σ, E), E' a set of equations such that SPEC |- E' and A in Beh(BSPEC). The **E'-realization** of A is the algebra

$$E'\text{-}R(A) = T_\Sigma(A_{Obs}) / \equiv_{eobs(A)+E'}$$

Proposition 4.6

Let BSPEC = (Obs, SPEC) be a behaviour specification with SPEC = (S, Σ, E), E' a set of equations such that SPEC |- E' and A in Beh(BSPEC).Then we have:

• The E'-realization of A is behaviourally equivalent to A

• The construction of this E'-realization extends to a functor E'-R: Beh(BSPEC) → Alg(SPEC*+E'), which is left adjoint to the forgetful functor U: Alg(SPEC*+E') → Beh(BSPEC), defined as the composition of the forgetful functors U1: Alg(SPEC*+E') → Alg(SPEC*) and U_{BSPEC}: Alg(SPEC*)→ Beh(BSPEC).

Definition 4.7

Let BSPECi = (Obsi, SPECi) with SPECi = (Si, Σi, Ei) i = 1, 2, be two behaviour specifications and h : BSPEC1 → BSPEC2 a view specification morphism. The functor $View_h$: Beh(BSPEC2) → Beh(BSPEC1), called **view functor associated to** h, is defined as the composition of the functors h(E1)-R : Beh(BSPEC2) → Alg(SPEC2*+h(E)) and U : Alg(SPEC2*+h(E))→ Beh(BSPEC), i. e. $View_h = U \circ h(E1)\text{-}R$.

In what follows, we will define the operation of view composition in terms of view specification

morphisms and present the usual correctness and compositionality theorems. However, the results have been harder to obtain since there is no amalgamated sum associated to pushout diagrams involving view morphisms. The only result were able to obtain and use has been a View Extension Lemma (see Appendix).

We could have provided a similar definition for view actualization, showing without problems the correctness of the operation. However, the standard prove of compositionality for the operation of actualization makes use of the amalgamation lemma that, as it was said above, does not hold for this kind of diagrams. However, it seems possible to prove compositionality using just the extension lemma.

With respect to union, there does not need to be a methodological justification of its definition via view morphisms. However, if there were we would have the same problems concerning the proof of compositionality

Definition 4.8

The view **composition** of two behaviour module specifications BMODj = (BPARj, BEXPj, BIMPj, BBODj, ej, sj, ij, vj), j = 1,2, via a pair h = (h1, h2) is defined in the same way as composition in 3.1 up to the following modifications:

- h1 and h2 are view specification morphisms
- behavioural consistency includes the property that e3 becomes a behaviour specification morphism

Theorem 4.8 (Correctness and compositionality of view composition)

Given correct behaviour module specifications BMOD1 and BMOD2 and a pair h = (h1, h2) of view specification morphisms such that the view composition BMOD3 = BMOD2 $*_h$ BMOD1 is defined and behavioural consistent, we have

1. **Correctness** BMOD3 is correct
2. **Compositionality** BSEM3 = BSEM1 $\cdot$ $View_{h1}$ $\cdot$ BSEM2
   where BSEMj, j = 1, 2, 3 is the behavioural semantics of BMODj.

Proof idea

See proof idea of theorem 3.2 using View Extension Lemma instead of Behaviour Extension Lemma.

5. References

[BEP 87] E.K. Blum, H. Ehrig, F. Parisi-Presicce: Algebraic Specification of Modules and their Basic Interconnections, *J. of Comp. Syst. Sci.* 34 (1987) 293-339.

[EFHLP 87] H. Ehrig, W. Fey, H. Hansen, M. Löwe, F. Parisi-presicce: Algebraic Theory of Modular Specification Development, Technical Report No. 87-06, Department of Computer Science, TU Berlin, September 1987.

[EM 85] H. Ehrig, B. Mahr, *Fundamentals of algebraic specification 1*, EATCS Monographs on Theor. Comp. Sc., Springer, 1985.

[EW 86] H. Ehrig, H. Weber: Programming in the Large with Algebraic Module Specifications, *Proc. IFIP Congress'86*, Dublin, Sept. 1986.

[GM 82] J.A. Goguen, J. Meseguer: Completeness of many-sorted equational logic, *Sigplan Notices* 17,1 (1982) 9-17.

[HW 85] R. Hennicker, M. Wirsing: Observational Specification: A Birkhoff-Theorem , *Recent Trends in Data Type Specification*, Informatik-Fachberichte, Springer 116 (1985) 119-135.

[MG 85] J. Meseguer, J. A. Goguen: Initiality, induction and computability, *Algebraic Methods in Semantics*, M. Nivat and J. Reynolds (eds.), Cambridge Univ. Press (1985) 459-540.

[NI 87] Mª P. Nivela: Semántica de Comportamiento en Lenguajes de Especificación, PhD. Thesis, Facultat d'Informàtica, Universitat Politècnica de Catalunya, Barcelona (1987).

[NO 87] Mª P. Nivela, F. Orejas: Initial Behaviour Semantics for Algebraic Specifications, to appear in *Proc. 5th Workshop on Algebraic Specifications of Abstract Data Types*, Gullane, 1987, Springer Lect. Notes in Comp. Sc.

[PP 87] F. Parisi-presicce: Union and Actualization of Module Specifications: Some Compatibility Results, *J. of Comp. Syst. Sci.* 35 (1987) 72-95.

[RE 81] H. Reichel: Behavioural equivalence - a unifying concept for initial and final specification methods, *Proc. 3rd Hungarian Computer Science Conf.*, Budapest (1981) 27-39.

[RE 84] H. Reichel: Behavioral validity of equations in abstract data types, *Contributions to General Algebra 3, Proc.of the Vienna Conference*, Verlag B. G. Teubner, Stuttgart (1985) 301-324.

[ST 85] D. Sannella: Tarlecki, A., On observational equivalence and algebraic specification *J J. of Comp. Syst. Sci.* 34 (2/3) (1987) 150-178.

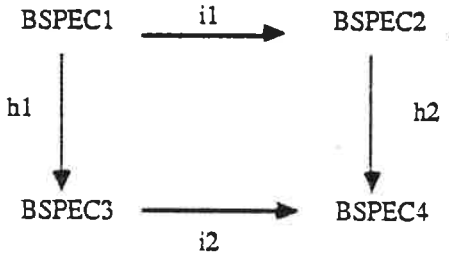
[SWi83] D. Sannella: Wirsing, M., A kernel language for algebraic specification and implementation. *Proc. Intl. Conf. on Foundations of Computation Theory* Sweden. Springer LNCS 158 (1983) 413-427.

[SC 82] O. Schoett: A theory of program modules, their specification and implementation (extended abstract), Univ. of Edinburgh, Rep. CSR-155-83, 1982.

Appendix

Definition 1 (Behavioural amalgamation)

Given the pushout diagram



where i1 and h1 are b-specification morphisms such that the pushout satisfies the observation preserving property, we define:

1. For all algebras A3∈ Beh(BSPEC3), A2∈ Beh(BSPEC2) and A1∈ Beh(BSPEC1) such that

$$BU_{h1}(A3) = A1 = BU_{i1}(A2)$$

the **behavioral amalgamated sum**, or short **b-amalgamation**, of A3 and A2 with respect to A1, denoted by $A4 = A3 \oplus_{A1} A2$, is the algebra in Beh(BSPEC4) defined as $A4 = A3 +_{A1} A2$ where $+_{A1}$ denotes the usual amalgamation in categories of algebras.

2. For all homomorphisms h3 : A3 → B3 in Beh(BSPEC3), h2 : A2 → B2 in Beh(BSPEC2) and h1: A1→ B1 in Beh(BSPEC1) with

$$BU_{h1}(f3) = f1 = BU_{i1}(f2)$$

the **behavioral amalgamated sum**, or short **b-amalgamation**, of f3 and f2 with respect to f1, denoted by $f4 = f3 \oplus_{f1} f2$, is the BSPEC4-behaviour morphism defined for every s in S4 as

$$f4_s = \textbf{if } s \in i3(S3) \textbf{ then } f3_s \textbf{ else } f2_s$$

where $+_{f1}$ denotes, as above, the usual amalgamation in categories of algebras.

Lemma 2 (Behavioural Amalgamation Lemma)

Given a pushout diagram as above b-amalgamation has the following properties:

1. Given algebras A3∈ Beh(BSPEC3), A2∈ Beh(BSPEC2) and A1∈ Beh(BSPEC1) such that

$$BU_{h1}(A3) = A1 = BU_{i1}(A2)$$

the b-amalgamation $A4 = A3 \oplus_{A1} A2$ is the unique algebra in Beh(BSPEC4), satisfying $A3 = U_{i2}(A4)$ and $A2 = U_{h2}(A4)$

Conversely, every $A4 \in$ Beh(BSPEC4) has a unique representation $A4 = A3 \oplus_{A1} A2$ where $A3 = BU_{i2}(A4)$, $A2 = BU_{h2}(A4)$ and $A1 = BU_{h1}(A3) = BU_{i1}(A2)$.

2 Given homomorphisms $h3 : A3 \rightarrow B3$ in Beh(BSPEC3), $h2 : A2 \rightarrow B2$ in Beh(BSPEC2) and $h1 : A1 \rightarrow B1$ in Beh(BSPEC1) with

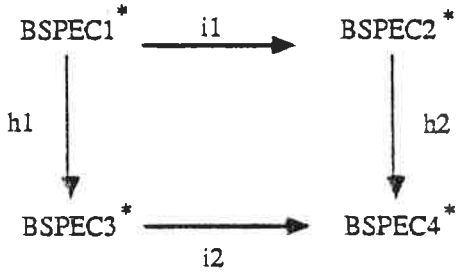
$$U_{h1}(f3) = f1 = U_{i1}(f2)$$

the b-amalgamation $f4 = f3 \oplus_{f1} f2$ is the unique homomorphism satisfying $f3 = BU_{i2}(f4)$ and $f2 = BU_{h2}(f4)$.

Conversely, every BSPEC4-behaviour morphism $f4 : A4 \rightarrow B4$ has a unique representation $f4 = f3 \oplus_{f1} f2$ where $f3 = BU_{i2}(f4)$, $f2 = BU_{h2}(f4)$ and $f1 = BU_{h1}(f3) = BU_{i1}(f2)$.

Proof idea

1. The key to this proof is showing that if if the pushout satisfies the observation preserving property this means that:



is also a pushout diagram. This is enough to show that for the given A1, A2 and A3, A4 belongs to Alg(BSPEC4*) and, therefore, to Beh(BSPEC4). The rest of the proof of this part is just straightforward.

2. In this part, again, the key point is showing that $f4 \in$ Beh(BSPEC4). This is done by induction on the structure of the $\Sigma 4$-observable computations and making use of the observation preserving property. []

Lemma 3 (Behaviour Extension Lemma)

1. Given a pushout diagram as above and a strongly persistent functor F: Beh(BSPEC1) → Beh(BSPEC2) there is a persistent functor F': Beh(BSPEC3) → Beh(BSPEC4) called **extension of F via h** and written F' = EXT(F,h) which is defined by the following b-amalgamations:

- $F'(A3) = A3 \oplus_{A1} F(A1)$ for every A3 in Beh(BSPEC3), where $A1 = BU_{h1}(A3)$
- $F'(f3) = f3 \oplus_{f1} F(f1)$ for every f3 in Beh(BSPEC3), where $f1 = BU_{h1}(f3)$

then, F' is uniquely defined (up to isomorphism) by $BU_{h2} \cdot F' = F \cdot BU_{h1}$.

2. If F is a free functor with respect to BU_{i1} then F' is free w.r.t. BU_{i2}.

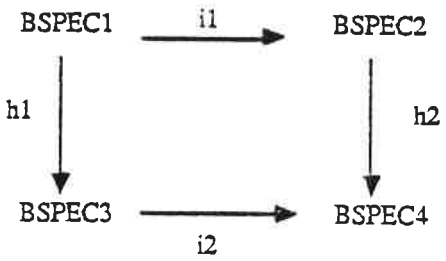
Proof idea

1. As usual, this part is a direct consequence of the (behaviour) amalgamation lemma.

2. Suppose B4∈Beh(BSPEC4) and f: A3 → BU_{i2}(B4) ∈Beh(BSPEC3), then let A1 = BU_{h1}(A3), B2 = BU_{h2}(B4) and f' = BU_{h1}(f). Since F_{i1} is free then there is a unique g': F(A1) → B2 in Beh(BSPEC2) such that BU_{i1}(g') = f'. Taking g = f $\oplus_{f'}$ g we have that g: F'(A3) → B4 and BU_{i2}(g) = f. Moreover, b-amalgamation lemma implies the uniqueness of g. []

Lemma 4 (View Extension Lemma)

Given the pushout diagram:



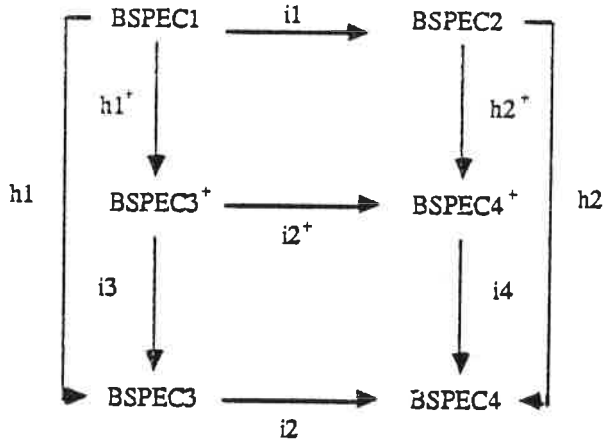
where h1 (and h2) are view behaviour specification morphisms, i1 and i2 are behaviour specification morphisms and the pushout satisfies the observation preserving property, we have that :

- If $BFree_{i1}$ is persistent then $BFree_{i2}$ is also persistent.
- Moreover, $BFree_{i2}$ is an extension of i1 in the following sense

$$BFree_{i1} \circ View_{h1} \equiv_{Sig2} View_{h2} \circ BFree_{i2}$$

Proof idea

The key point in this proof is the decomposition of the given diagram into



where $BSPEC3^+$ = (Obs3+h1(Obs1), SP3) and $BSPEC4^+$ = (Obs4+h2(Obs2), SP4), $h1^+$ and $h2^+$ are defined (on sorts and operations) as h1 and h2, and i3 and i4 are the identity (on sorts and operations) morphisms. Then, persistency of $BFree_{i1}$ implies persistency of $BFree_{i2^+}$. Also, persistency of $BFree_{i2^+}$ implies persistency of $BFree_{i2}$ since the former can be seen as a realization of the latter.
On the other hand, again by the Behaviour Extension Lemma, $BFree_{i2^+}$ is an extension of $BFree_{i1}$. To prove

$$BFree_{i2^+} \circ View_{i3} \equiv_{Sig2^+} View_{i4} \circ BFree_{i2}$$

it is enough to take into account that the view functors are, in this case, free functors associated to the inverse morphisms $i3^{-1}$ and $i4^{-1}$ which are observable sort preserving although not necessarily non-observable sort preserving (cf. remark 2.1.2.2). Then the composition of free functors is also free and, therefore, $BFree_{i2^+} \circ View_{i3}$ and $View_{i4} \circ BFree_{i2}$ are naturally isomorphic, which implies

$$BFree_{i1} \circ View_{h1} \equiv_{Sig2} View_{h2} \circ BFree_{i2}$$

[]

It must be noted that for this kind of diagrams amalgamation cannot be defined in a way that satisfies the amalgamation lemma. The problem consists in that if we have a view behaviour specification morphism h: BSPEC1 → BSPEC2 then, for a given A2 in BSPEC2, $View_h(A2)$ may be not behaviourally equivalent to $BU_h(A2)$. This happens, for instance, if

bspec BSPEC1 = sorts obs s1	bspec BSPEC2 = sorts non obs s1
ops a, b : → s1	opns a, b : → s1
end bespec	end bspec

h is the obvious inclusion, and A2 is the algebra $A2_{s1} = \{v\}$, $a_{A2} = b_{A2} = v$, then $BU_h(A2_{s1}) = \{v\}$ while $View_h(A2) = \{a,b\}$.

This means that amalgamation, satisfying the lemma, cannot exist since any reasonable definition of

$$A4 = A3 \oplus_{A1} A2$$

in the diagram above, should satisfy $BU_{h2}(A4) = A2$ and, on the other hand, an adequate version of the amalgamation lemma would ask for $View_{h2}(A4) \equiv_{Sig2} A2$.