Constraints for behavioural specifications

Fernando Orejas
M. Pilar Nivela

Report LSI–90–35

Constraints for Behavioural Specifications

F. Orejas, M. P. Nivela
Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

Behavioural specifications with constraints for the incremental development of algebraic specifications are presented. The behavioural contraints correspond to the completely defined subparts of a given incomplete behavioural specification. Moreover, the *local* observability criteria used within a behavioural constraint could not coincide with the global criteria used in the behavioural specification. This is absolutely needed because, otherwise, some constraints could involve only non observable sorts and therefore have trivial semantics. Finally, the extension operations and completion operations for refining specifications are defined. The extension operations correspond to horizontal refinements and build larger specifications on top of existing ones in a conservative way. The completion operations correspond to vertical refinements, they *add detail* to an incomplete behavioural specification and they do restrict the class of models.

1. Introduction

A formal framework for the incremental development of algebraic specifications is presented in [OSC 89]. The main ideas of this approach are:

1. The possibility of dealing with *incomplete* specifications at any stage of the development process. *Incompleteness* means that there may be not enough equations for defining the operations of the specification or there may be not enough operations to generate all the *values* of a certain sort. It is our believe that any approach for formalizing the specification development process from informal requirements should be capable of dealing with such kind of incomplete specifications. The reason is that, on the one hand, informal requirements are usually incomplete (even inconsistent) and, as a consequence, the specifier must take *design* decisions within the development process that would make the final specification complete and consistent. On the other hand, prematurely taking this decisions may cause severe problems if it is later discovered that these decisions were inadequate from the customer point of view. This may mean in practice that all the work done since the inadequate decision could be wasted. The way of handling this incompleteness in this approach was by means of algebraic specifications with constraints [Rei 80, BG 80]. The constraints correspond to the completely defined subparts of a given incomplete specification. The corresponding semantics is then loose, accepting as models all algebras satisfying the axioms and all the constraints of the specification. A related approache in this context is the pioneering concept of *canon* [Rei 80] which essentially coincides with

our notion of incomplete specification but allowing also to deal with partial operations and algebras. Even more related than the work of Reichel is the work on the design of the specification language Look [ETLZ 82], in which many technical and methodological ideas of [OSC 89] could be found. However, the results obtained in [OSC 89] go beyond the ones used for the semantic definition of Look and, in fact, some open problems were solved.

2. Related with the notion of incomplete specification is the idea of developing specifications by means of horizontal and vertical of refinements. In more classical approaches in the field of algebraic specification (e.g. [GB 80]) the specifications are developed only by horizontal refinements (i.e. extensions), while vertical refinements were considered only for the development of implementations. In our context, vertical refinements are the operations by which we *add detail* to an incomplete specification, i.e. vertical refinements make the specifications more complete. At the semantic level this is seen as a restriction on the class of models. Our notion of vertical refinement coincides with the notion used by Sannella and Wirsing and Sannella and Tarlecki to define implementations [SW 83, ST 87a, ST 87b], even though the aims are different because they are more interested with the development of programs from specifications. In fact, most of the methodological ideas underlying our approach and theirs are the same. However, there is a fundamental differece in the sense that they are only concerned with what happens at the model level and never try to obtain compatibility results or even to describe their ideas at the specification level. In this sense, our approach can be considered an extension and a complement of theirs, in that one of our main aims is to obtain this kind of compatibility results. We can also say that, in some sense, our methodological ideas about the incremental development of specifications may be found in the specification language Larch (the connection to Look has already been established). However the lack of precise formal semantics (to our knowledge) make difficult a comparison at the technical level. Anyhow, our approach could be seen as providing the adequate framework for writing such a semantic definition.

3. The way of handling incomplete specifications and the *interaction* of horizontal and vertical refinements make useless, in our approach, the use of (explicitly) parameterized specifications. The reason is that every incomplete specification may be seen as implicitly parameterized by its incomplete subspecifications. In particular, the abovementioned interaction of horizontal and vertical refinement allows to substitute any incomplete subspecification of a given specification by a more complete one in a way that generalizes parameter passing in the more standard approaches [EM 85]. In fact, the results obtained in our approach generalize all classical results on parameter passing by just requiring a limited form of persistency.

Being convinced that the notions of behaviour and observability are critical with respect to the semantics of software specifications, from the very beginning we wanted to express all the framework in the behaviour setting defined in [Niv 87, NO 88] (for related approaches to behavioural specifications see e.g. [HW 85, MG 85, ST 87a, Rei 81]). However there seemed to be a technical problem: in the standard case most of the results and semantics constructions were obtained making heavy use of the Amalgamation Lemma for specifications with constraints [Ehr 89] but, on the other hand, in [ONE 89] it was shown that Amalgamation Lemma was only possible (under certain

reasonable restrictions) for pushout diagrams involving behavioural specification morphisms, but not when involving the so called *view specification morphisms*. Now the problem was that because of the need of having different observability criteria within the same specification (certain sorts are considered non-observable at the global level but may be considered locally observable within a constraint) there was a need of dealing with this view morphisms that would cause all the troubles.

Fortunately, we were able to provide the adequate definitions, both from the methodological (we think) and from the technical point of view, that would allow us to obtain all the needed results. To do that, we had to generalize the Amalgamation and Extension Lemmas for behaviour specification morphisms and the Extension Lemma for view morphisms for the case of specifications with constraints. Also, we had to develop a very restricted version of the Amalgamation for view morphisms that would only apply to free algebras. But, once this was done, must of the proofs and constructions from [OSC 89] could be directly translated to the new setting, with some exception in which a use of the Amalgamation Lemma in [OSC 89] was now translated into the use of the Extension Lemma for view morphisms. This experience apparently showed that, in fact, the whole approach could be parameterized being independent of any arbitrary Specification Logic or Institution [GB 85], as long as a reasonble amount of basic constructions (amalgamation and extensions) are provided. In this sense, we think that this could be done by extending some preliminary results that were presented in [EPO 89].

Most of the related work (that we know) to our framework has already been mentioned: it mainly has to do with the *standard* setting as defined in [OSC 89]. With respect to the new aspects presented in this paper, i.e. the handling of behavioural constraints, the only related work we know is from Reichel [Rei 87]. However there are big differences between the two approaches not only in the aims, since the kind of results we obtain are of different nature of the ones obtained by him, but also technical in two senses: a) our notion of behavioural equivalence is stronger, since algebras that only differ on non-observable junk would be not equivalent for him but they would be for us, b) on the very notion of constrained specification because, according to his approach, observability in a behavioural canon is global, i.e. a sort can either be observable or non-observable in the whole specification while, for us, a sort may be non observable at the global level but may be considered observable locally within a constraint. The reason for this is that, otherwise, some constraints could involve only non-observable sorts and therefore have trivial semantics.

The organization of the paper is as follows: in the next section we provide the basic definitions and notation about behavioural specifications. In the third section we present the main basic tools to be used for proving all the results: we provide the Amalgamation and Extension Lemmas for behavioural specifications slightly generalized with respect to the version of [OSC 89]. Also, we give the restricted version of the Amalgamation Lemma for the view case that was mentioned above. In section 4, we define our concept of behavioural specification with constraints and we specialize some results of the previous section to this setting. The operations for refining specifications are defined and the main results are obtained in section 5. In section 6 some conclusions are presented.

2. Behavioural Semantics

In this section a summary of the behavioural framework is given. For more details see [Niv 87, NO 88, ONE 89].

2.1 Basic Behavioural Concepts

Given a signature $\Sigma = (S, \Omega)$ a **behaviour signature** $B\Sigma$ is a triple $B\Sigma = (Obs, S, \Omega)$ with $Obs \subseteq S$. The sorts in Obs are called **observable sorts**. A behaviour signature determines a set of observable computations which will provide its observable behaviour. A **computation** is a term in $T_\Sigma(X_{Obs})$ where $X_{Obs} = \{X_s\}_{s \in Obs}$ is a family of observable variables. A computation of observable sort, that is, in $T_\Sigma(X_{Obs})_s$ with $s \in Obs$, is called an **observable computation**. Analogously, a **computation over a Σ-algebra** A is a term in $T_\Sigma(A_{Obs})$. A computation of observable sort, that is, in $T_\Sigma(A_{Obs})_s$ with $s \in Obs$, is called an **observable computation over** A.

We may associate two categories of models to every behaviour signature $B\Sigma$: the well-known category $Alg(\Sigma)$ of Σ-algebras and Σ-homomorphisms, and the category $Beh(B\Sigma)$ which defines behavioural semantics. In this category objects are Σ-algebras as in $Alg(\Sigma)$ but morphisms are different. To avoid confusion from now on morphisms in $Alg(\Sigma)$ will be called Σ-homomorphisms while morphisms in $Beh(B\Sigma)$ will be called Σ-behaviour morphisms.

A **Σ-behaviour morphism** $f: A \to B$ between two Σ-algebras A and B is an Obs-indexed family of mappings $f = \{f_s\}_{s \in Obs}$ preserving all the observable computations, that is, for every $t \in T_\Sigma(A_{Obs})_s$, $s \in Obs$, it holds that $f_s(\varepsilon_A(t)) = \varepsilon_B(f^\#_s(t))$ where $f^\#: T_\Sigma(A_{Obs}) \to T_\Sigma(B_{Obs})$ is the unique Σ-homomorphism which extends f and ε_A is the evaluation of terms in A, i.e. the unique Σ-homomorphism extending the inclusion of A_{Obs} into A. Σ-algebras together with Σ-behaviour morphisms form the category **Beh(BΣ)**.

If Obs coincides with S then $Beh(B\Sigma)$ is exactly the same as $Alg(\Sigma)$ and if there are no observable sorts in Σ then Σ-behaviour homomorphisms are empty sets.

A Σ-behaviour morphism f establishes a relationship between the observable computations t over A and $f^\#(t)$ over B in such a way that it is compatible with their results $\varepsilon_A(t)$ in A and $\varepsilon_B(f^\#(t))$ in B respectively. Thus an Obs-indexed family $f = \{f_s\}_{s \in S}$ is a Σ behaviour morphism if these observable computations over A yield in B the same value as in A, up to the transformation determined by f. 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 $Beh(B\Sigma)$. In particular, isomorphism in $Beh(B\Sigma)$ coincides with the notion of behavioural equivalence from [MG 85, HW 85, SW 83, ST 85].

A Σ-behaviour morphism f such that f_s is bijective for every s in Obs is a Σ-behaviour isomorphism in the category $Beh(B\Sigma)$.

Two Σ-algebras A and B are **behaviourally equivalent**, denoted $A \equiv_{B\Sigma} B$, if there exists a Σ-behaviour isomorphism $f: A \to B$ between them. Behavioural equivalence is an equivalence relation between Σ-algebras and every equivalence class is called a **behaviour**.

A **Σ-context over the sort** s is a term $c[z] \in T_\Sigma(X_{Obs} \cup \{z\})_{s'}$ with $s' \in Obs$ and sort(z) = s. By c[t] we denote the **application of the context over** t, that is, $\bar{\sigma}(c[z])$ where σ is the assigment $\sigma: X_{Obs} \rightarrow T_\Sigma(X_{Obs})$ defined by σ(z) = t and σ(x) = x for every x in X_{Obs}. Analogously, a **Σ-context over the sort s for a** Σ-algebra A is a term $c_A[z] \in T_\Sigma(A_{Obs} \cup \{z\})_{s'}$ with $s' \in Obs$ and sort(z) = s.

A Σ-algebra A **behaviourally satisfies** the Σ-equation *e*: $\lambda Y.t_1 = t_2$, denoted by $A \models_B e$, if A satisfies $\lambda X_{Obs}.c[\bar{\sigma}(t_1)] = c[\bar{\sigma}(t_2)]$ for every Σ-context c[z] over the sort of *e* and every assignment $\sigma: Y \rightarrow T_\Sigma(X_{Obs})$.

A **behaviour presentation** BP is a 4-tuple, BP = (Obs, S, Ω, E) where BΣ = (Obs, S, Ω) is a behaviour signature and E a set of Σ-equations. **Beh(BP)** is the full subcategory of Beh(BΣ) of all Σ-algebras which behaviourally satisfy the equations in E. In what follows we will also denote BP by BP = (Obs, P) with P = (S, Ω, E), Obs ⊆ S and Σ = (S, Ω), where P is called a **presentation**. We will indistinctly write $A \equiv_{B\Sigma} B$ or $A \equiv_{BP} B$.

2.2 Presentation morphisms and their associated functors

The relationships that can be established between two behaviour presentations BP1 = (Obs1, P1) and BP2 = (Obs2, P2) are as usual defined by *presentation morphisms* h : BP1 → BP2, that is, a signature morphism h : Σ1 → Σ2 such that E2 |- h(E1). But now it is necessary to make the relationship between the observability criteria of BP1 and BP2 explicit. If the observable sorts are preserved, then h is said to be a *weak presentation morphism*. If the non observable sorts are preserved then h is called a *view presentation morphism*. Finally, a *behaviour presentation morphism* preserves both the observable and the non observable sorts.

Definition 2.2.1

Let BP1 = (Obs1, P1) and BP2 = (Obs2, P2) be two behaviour presentations and h: P1 → P2 a presentation morphism. We say that h: BP1 → BP2 is a

a) **weak presentation morphism** if h(Obs1) ⊆ Obs2

b) **view presentation morphism** if h(S1-Obs1) ⊆ S2-Obs2

c) **behaviour presentation morphism** if h(Obs1) ⊆ Obs2 and h(S1-Obs1) ⊆ S2-Obs2

The associated categories are the following:

a) **Weak-BP** is the category of behaviour presentations and weak presentation morphisms. The usual pushout constructions in **P** can be extended in a simple way to pushouts in **Weak-BP**.

b) **BP** is the category of behaviour presentations and behaviour presentation morphisms. Obviously, pushouts in the category **BP** are defined in the same way as in **Weak-BP**.

c) **View-BP** is the category of behaviour presentations and view presentation morphisms. Pushout constructions are easily obtained by using the ones of the non observable sorts (in the category of **Sets**).

For every weak presentation morphism h: BP1 → BP2 (resp. behaviour presentation

morphism) there is a forgetful functor BU_h: Beh(BP2) → Beh(BP1) defined as usual.

Every weak presentation morphism h (resp. behaviour presentation morphism) has an associated free functor $BFree_h$, which is left adjoint to the forgetful functor BU_h, and is defined by

$$BFree_h(A) = T_{\Sigma 2}(A_{Obs1}) / \equiv_{h(obs\text{-}eq(A))+E2}$$

where the values $a \in A_s$ are interpreted as values of sort h(s).

If Obs1 = S1 then $BFree_h(A)$ is the usual free construction for every P1-algebra A.

The behavioural equivalence relation may be extended uniformly from algebras to functors, that is, behavioural equivalence of functors coincides with natural isomorphism. If F and F' are two functors from Beh(BP1) to Beh(BP2) we will say that F and F' are behaviourally equivalent if they are naturally isomorphic, which will be denoted by $F \equiv F'$. Therefore, we immediately have that any functor behaviourally equivalent to a free functor is also free.

However, if h is a view presentation morphism then it has no associated forgetful functor. The reason is that there can be less observable sorts in Obs2 than in Obs1. This means that when *forgetting* over a BP2-behaviour morphism $f = \{f_s : A2_s \rightarrow A2'_s\}_{s \in Obs2}$ there can exist some sort $s \in Obs1$ such that $h(s) \notin Obs2$ and therefore $f_{h(s)}$ would not be defined. Passing from BP2 to BP1 behaviours can be done by a functor $View_h$, called *view functor*, which builts up a BP1-behaviour morphism from a BP2-behaviour morphism and describes how the models of Beh(BP2) are *seen* from the BP1 *point of view*. First of all a special realization of the behaviour of an algebra A2 in Beh(BP2) is constructed. This realization belongs to the category $Alg(P2^+)$ and is behaviourally equivalent to A2, in in such a way that BP2-behaviour morphisms can be extended to usual $P2^+$ homomorphisms. After that a forgetful functor from $Alg(P2^+)$ to Beh(BP1) is applied to this realization.

Definition 2.2.2

Let BP = (Obs, P) with P = (S, Ω, E) be a behaviour presentation. The presentation P^* **behaviourally derived from** BP is defined as $P^* = (S, \Omega, E^*)$ where E^* is the set all observable properties deduced from E, that is, $E^* = \{t_1 = t_2 \mid t_1, t_2 \in T_\Sigma(X_{Obs})_s, s \in Obs, E \vdash t_1 = t_2\}$.

A Σ-algebra A belongs to Beh(BP) if and only if A belongs to $Alg(P^*)$.

Definition 2.2.3

Let h: BP1 → BP2 be a view presentation morphism, $BP2^+$ the behaviour presentation given by $BP2^+ = (Obs2 + h(Obs1), P2^+)$ with $P^+ = P^* + (\emptyset, \emptyset, h(E1))$. Let h^+: BP1 → $BP2^+$ be the behaviour presentation morphism defined as h on sorts and operations, and let η be the weak presentation inclusion η: $BP2^* \rightarrow BP2^+$.

The h(E1)-realization functor $R_{h(E1)}$ is defined by the composition of functors $R_{h(E1)} = BFree_\eta \circ Id$, where Id is the identity functor between the categories Beh(BP2) and $Beh(BP2^*)$.

Proposition 2.2.4

$R_{h(E1)}(A2)$ is behaviourally equivalent to A2 for every algebra A2 in Beh(BP2) .

Definition 2.2.5

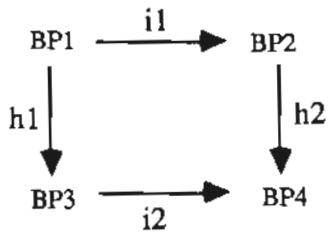
Let BP1 = (Obs1, P1) and BP2 = (Obs2, P2) be two behaviour presentations with P1 = (S1, Ω1, E1) and P2 = (S2, Ω2, E2). Let h: BP1 → BP2 be a view presentation morphism.

The functor $View_h$: Beh(BP2) → Beh(BP1), called **view functor associated to** h, is defined as $View_h = BU_{h^+} \circ R_{h(E1)}$

2.3 Pushout constructions

When putting together two behaviours BP2 and BP3 with a common sub-behaviour BP1 it may happen that the resulting behaviour BP4 is not the right combination of BP1 and BP2 behaviours because the observable computations of BP2 and BP3 may be combined to cause side effects in the observable computations of BP4.

This means that not every pushout diagram {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} of the form



with the involved presentation morphisms being any of the previous three kinds, will be useful when dealing with behavioural semantics. This kind of discontinuity, if allowed, originates several undesired effects being the most important one the incompatibility of the semantic constructs used at the presentation and model levels. This problem is overcome if the pushout satisfies the *observation preserving property*. We say that an observable computation $t \in T_\Sigma(X_{Obs})_{Obs}$ is **minimal** if no subterm of t different from a variable is an observable computation.

Definition 2.3.1

A pushout diagram {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} in the category **BP** or in **Weak-BP** satisfies the **observation preserving property** if, for any set X_{Obs4} of observable variables, for every minimal observable computation $t \in T_{\Sigma 4}(X_{Obs4})$ and for every $s \in S4 - Obs4$ being the sort of a non observable subterm of t, it holds that $s \in S4\text{-}i2 \circ h1(S1)$.

If a pushout satisfies the observation preserving property then every minimal observable computation t belongs either to $T_{h2(\Sigma 2)}(X_{Obs2})$ or to $T_{i2(\Sigma 3)}(X_{Obs3})$.

When dealing with the category **View-BP** we need a slightly different version of the observation preserving property, as we will see in the proof of the existence of the extension lemma.

Definition 2.3.2

A pushout diagram {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} in the category **View-BP** with i1, i2 behaviour presentation morphisms of **BP**, satisfies the **observation preserving property** if the diagram $\{P4^+, i2^+, h2^+\}$ = po $\{P1, P2, P3^+, i1, h1^+\}$ satisfies the observation preserving property in **BP**.

So, also in this case, every minimal observable computation $t \in T_{\Sigma 4}(X_{Obs4})$ belongs either to $T_{h2(\Sigma 2)}(X_{Obs2})$ or to $T_{i2(\Sigma 3)}(X_{Obs3})$.

The observation preserving property of all the pushouts diagrams in this paper is assumed. For this reason we will not explicitly state this porperty.

3. Behavioural Amalgamation and Extension properties

This section describes the amalgamation and extension properties that can be obtained in each of the categories **BP** and **View-BP** (unfortunately, in the category **Weak-BP** in general there are neither Amalgamation nor Extension Lemmas). First, we will state the Amalgamation Lemma associated to the category **BP** (its proof can be found in [ONE 89]). Then we will see a slight generalization, with respect to [ONE 89], of the Extension Lemmas associated to the categories **BP** and **View-BP**. Finally, we will present a restrictive (with respect to free algebras) version of the Amalgamation Lemma for **View-BP**. This restrictive version is caused by the problem that, in general, in **View-BP** there are no appropriate amalgamated sums. Nevertheless, that restrictive version is sufficient for our purposes in the following sections.

Definition 3.1 (Behavioural Amalgamation)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1}be pushout diagram in **BP**.

1. For all algebras A3 ∈ Beh(BP3), A2 ∈ Beh(BP2) and A1 ∈ Beh(BP1) such that $BU_{h1}(A3) = A1 = BU_{i1}(A2)$ the **behavioral amalgamated sum** of A3 and A2 with respect to A1, denoted by $A4 = A3 +_{A1} A2$, is the algebra in Beh(BP4) defined by $A4 = A3 +_{A1} A2$ where $+_{A1}$ denotes the usual amalgamation in categories of algebras.

2. For all behaviour morphisms h3: A3 → B3 in Beh(BP3), h2: A2 → B2 in Beh(BP2) and h1: A1 → B1 in Beh(BP1) with $BU_{h1}(f3) = f1 = BU_{i1}(f2)$ the **behavioral amalgamated sum** of f3 and f2 with respect to f1, denoted by $f4 = f3 +_{f1} f2$, is the BP4-behaviour morphism defined for every s in S4 as $f4_s$ = **if** s ∈ i2(S3) **then** $f3_s$ **else** $f2_s$.

Lemma 3.2 (Behavioural Amalgamation Lemma)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} be pushout diagram in **BP**.

1. Given algebras A3∈Beh(BP3), A2∈Beh(BP2) and A1∈Beh(BP1) such that $BU_{h1}(A3) = A1 = BU_{i1}(A2)$ the behavioural amalgamation $A4 = A3 +_{A1} A2$ is the unique algebra in Beh(BP4) which satisfies $A3 = BU_{i2}(A4)$ and $A2 = BU_{h2}(A4)$.

Conversely, every A4 ∈ Beh(BP4) has a unique representation $A4 = A3 +_{A1} A2$ where $A3 = BU_{i2}(A4)$, $A2 = BU_{h2}(A4)$ and $A1 = BU_{h1}(A3) = BU_{i1}(A2)$.

2 Given behaviour morphisms h3: A3 → B3 in Beh(BP3), h2: A2 → B2 in Beh(BP2) and h1: A1→ B1 in Beh(BP1) with $BU_{h1}(f3) = f1 = BU_{i1}(f2)$ the behavioural amalgamation $f4 = f3 +_{f1} f2$ is the unique homomorphism satisfying $f3 = BU_{i2}(f4)$ and $f2 = BU_{h2}(f4)$.

Conversely, every BP4-behaviour morphism f4: A4 → B4 has a unique representation $f4 = f3 +_{f1} f2$ where $f3 = BU_{i2}(f4)$, $f2 = BU_{h2}(f4)$ and $f1 = BU_{h1}(f3) = BU_{i1}(f2)$.

Definition 3.3

Let BP1, BP2 be two behaviour presentations and h: BP1 → BP2 a (weak) behaviour presentation morphism and let A be a subcategory of Beh(BP1).

A functor G: Beh(BP1) → Beh(BP2) is (strongly) persistent relative to A iff for every A in A $A = BU_h(G(A))$.

Lemma 3.4 (Behaviour Extension Lemma)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} be a pushout diagram in **BP**. Let A3 and A1 be subcategories of Beh(BP3) and Beh(BP1) respectively such that $BU_{h1}(A3)$ is included in A1. Finally, let F: Beh(BP1) → Beh(BP2) be a strongly persistent functor relative to A1.

1. There exists a unique (up to isomorphism) persistent relative to A3 functor F': A3 → Beh(BP4) such that $BU_{h2} \circ F' = F \circ BU_{h1}$ which moreover is defined by

   (i) $F'(A3) = A3 +_{A1} F(A1)$ for every A3 in A3 with $A1 = BU_{h1}(A3)$
   (ii) $F'(f3) = f3 +_{f1} F(f1)$ for every f3 in A3 with $f1 = BU_{h1}(f3)$

2. If F restricted to A1 is a free functor with respect to BU_{i1} then F' is free w.r.t. BU_{i2}.

Proof Sketch

1. Trivially, F' is a functor as defined by i) and ii) and, by construction, is persistent relative to A3 and is an extension of F.

2. Suppose B4∈ Beh(BSPEC4) and f: A3 → $BU_{i2}(B4)$ ∈ A3, 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 $g = f +_{f'} g$ we have that g: F'(A3) → B4 and $BU_{i2}(g) = f$. Moreover, the Behavioural Amalgamation Lemma implies the uniqueness of g. []

It is not always possible to define amalgamated sums for pushouts in the category **View-BP**. For instance, consider the following behaviour presentations

bpres BP1 = **obs sorts** s1, s2 **ops** a: → s1 **end bpres**
bpres BP2 = **obs sorts** s1, s2, s3 **ops** a: → s1, g: s1 s3 → s2 **end bpres**
bpres BP3 = **obs sorts** s1 **non obs sorts** s2 **ops** a: → s1 **end bpres**
bpres BP4 = **obs sorts** s1, s3 **non obs sorts** s2 **ops** a: → s1, g: s1 s3 → s2 **end bpres**

The algebra A4 = { $\{a\}_{s1}$, $\{b\}_{s3}$, $\{g(a, b)\}_{s2}$ } cannot be properly decomposed as an amalgmated sum. The algebras A2 and A3 should be defined by $A2 = View_{h2}(A4) = \{ \{a\}_{s1}, \{b\}_{s3},$

$\{g(a, b)\}_{s2}$ } and A3 = $BU_{i2}(A4)$ = { $\{a\}_{s1}$, $\{g(a, b)\}_{s2}$ }. But then $View_{h1}(A3)$ = { $\{a\}_{s1}$, $\emptyset_{s2}$ } and $BU_{i1}(A2)$ = { $\{a\}_{s1}$, $\{g(a, b)\}_{s2}$ } which are not equal.

However, in the view case we have a restricted version of the Extension Lemma that will allow us to express an amalgamation descomposition for the subclass of algebras which are free constructions.

Lemma 3.5 (View Extension Lemma)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1}be pushout diagram in **View-BP** with i1and i2 behaviour presentation morphisms. Let A3 and A1 be subcategories of Beh(BP3) and Beh(BP1) respectively such that $View_{h1}(A3)$ is included in A1.

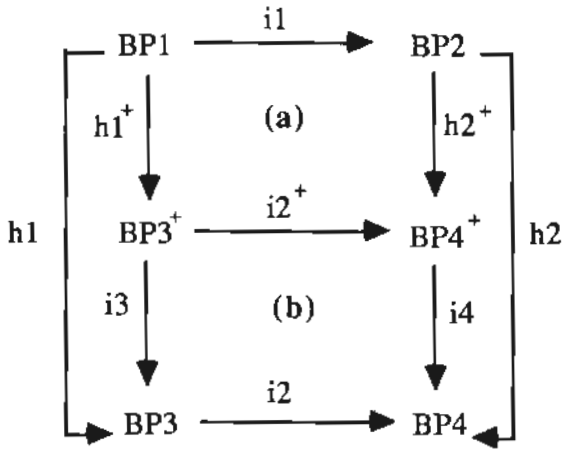
If $BFree_{i1}$ is persistent relative to A1 then

(i) $BFree_{i2}$ is persistent relative to A3

(ii) $BFree_{i2}$, with respect to algebras is A3, is an extension of $BFree_{i1}$, that is, $BFree_{i1} \circ View_{h1}(A3) \equiv_{BP2} View_{h2} \circ BFree_{i2}(A3)$, for every A3 in A3.

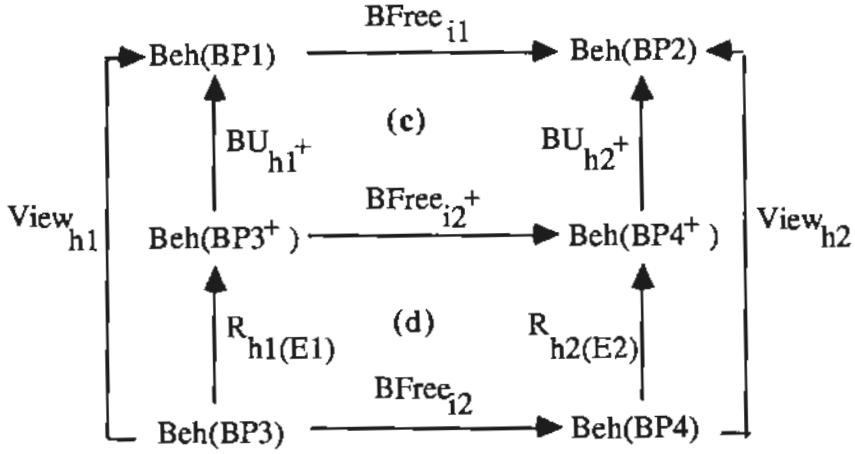
Proof

We can consider the following presentation diagram (1)



where $BP3^+$ is the presentation $BP3^+$ = (Obs3+h1(Obs1), $P3^*$+h1(E1)), $BP4^+$ is the presentation $BP4^+$ = (Obs4+h2(Obs2), $P4^*$+h2(E2)), $h1^+$ and $h2^+$ are defined (on sorts and operations) as h1 and h2, and i3 and i4 are the inclusion morphisms i3: $BP3^* \rightarrow BP3^+$, i4: $BP4^* \rightarrow BP4^+$.

Its corresponding semantic diagram (2) is



The functor $BFree_{i2}+$ is an extension of $BFree_{i1}$ by the Behaviour Extension Lemma (applied to the subdiagram **(a)** which is a pushout in **BP**). Moreover, $R_{h1(E1)}$ is in fact a free functor. Since the composition of free functors is also a free functor we have that $BFree_{i2}+ \circ R_{h1(E1)}$ and $R_{h2(E2)} \circ BFree_{i2}$ are naturally isomorphic and therefore

$$BFree_{i1} \circ View_{h1} \equiv_{BP2} View_{h2(E2)} \circ BFree_{i2}$$

The relative persistency of $BFree_{i1}$ implies the relative persistency of $BFree_{i2}+$ by the Behaviour Extension Lemma. Moreover, the relative persistency of $BFree_{i2}+$ implies the relative persistency of $BFree_{i2}$ since the former can be seen as a realization of the latter. []

By having this version of the Extension Lemma, it is possible to represent (up to behavioural equivalence) the subclass algebras in Beh(BP4) which are free constructions over algebras of Beh(BP3) as amalgamated sums of appropriate algebras in Beh(BP3) and Beh(BP2).

Definition 3.6 (View Amalgamation of algebras in the free case)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1}be a pushout diagram in **View-BP** with i1and i2 behaviour presentation morphisms. Let A1, A3 and A4 be subcategories of Beh(BP1), Beh(BP3) and Beh(BP4) respectively such that $View_{h1}(A3) = A1$, $BU_{i2}(A4) = A3$ and $View_{h2}(A4) = Free_{i1}(A1)$. Let us also suppose that $BFree_{i1}$ is strongly persistent relative to A1.

For all algebras $A3 \in A3$, $A2 \in$ Beh(BP2) and $A1 \in A1$ such that

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

(ii) $A2 = BFree_{i1}(A1)$

the **view amalgamated sum** of A3 and A2 with respect to A1, denoted by $A4 = A3 \oplus_{A1} A2$, is the algebra in Beh(BP4) defined as $A4 = R_{h1(E1)}(A3) +_{A1} A2$ where $+_{A1}$ denotes behaviour amalgamation.

To see that this definition has sense let us consider the above presentation diagrams (1) and (2). The algebra $R_{h1(E1)}$ is in Beh($BP3^+$) and satisfies $BU_{h1}+(A3) = A1 = BU_{i1}(A2)$. Therefore $A4 = R_{h1(E1)}(A3) +_{A1} A2$ is an algebra in Beh($BP4^+$) and also in Beh(BP4).

The same argument allows to define the amalgamation of behaviour morphisms as stated in the following definition.

Definition 3.7 (View Amalgamation of morphisms in the free case)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1}be pushout diagram in **View-BP** with i1 and i2 behaviour presentation morphisms. Let A1, A3 and A4 be subcategories of Beh(BP1), Beh(BP3) and Beh(BP4) respectively such that $View_{h1}(A3) = A1$, $BU_{i2}(A4) = A3$ and $View_{h2}(A4) = Free_{i1}(A1)$ Let us also suppose that $BFree_{i1}$ is strongly persistent relative to A1.

For all behaviour morphisms h3: A3 → B3 in Beh(BP3), h2: A2 → B2 in Beh(BP2) and h1: A1 → B1 in Beh(BP1) such that

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

(ii) $f2 = BFree_{i1}(f1)$

the **view amalgamated sum** of f3 and f2 with respect to f1, denoted by $f4 = f3 \oplus_{f1} f2$, is the BP4-behaviour morphism defined by $f4 = R_{h1(E1)}(f3) +_{f1} f2$.

Lemma 3.8 (View Amalgamation Lemma in the free case)

Let {BP4, i2, h2} = po {BP1, BP2, BP3, i1, h1} be pushout diagram in **View-BP** with i1and i2 behaviour presentation morphisms. Let A1, A3 and A4 be subcategories of Beh(BP1), Beh(BP3) and Beh(BP4) respectively such that $View_{h1}(A3) = A1$, $BU_{i2}(A4) = A3$ and $View_{h2}(A4) = Free_{i1}(A1)$. Moreover, assume that $Free_{i1}$ is persistent relative to A1. Then:

1. Given algebras A3 ∈ A3, A2 ∈ Beh(BP2) and A1 ∈ A1 such that

(i) $A2 = BFree_{i1}(A1)$

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

the view amalgamation sum $A4 = A3 \oplus_{A1} A2$ is a free construction w.r.t. A3.

Moreover, for every A4 ∈ Beh(BP4) it holds that $A4 \equiv_{BP4} R_{h1(E1)}(A3) +_{A1} A2$, where $A3 = BU_{i2}(A4)$, $A2 = View_{h2}(A4)$ and $A1 = View_{h1}(A3) = BU_{i1}(A2)$.

2 Given behaviour morphisms h3: A3 → B3 in A3, h2: A2 → B2 in Beh(BP2) and h1: A1→ B1 in A1 with

(i) $f2 = BFree_{i1}(f1)$

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

the view amalgamation sum $f4 = f3 \oplus_{f1} f2$ satisfies that $f4 = Free_{i2}(f3)$.

Moreover, every BP4-behaviour morphism f4: A4 → B4 is naturally isomorphic to $R_{h1(E1)}(f3) +_{f1} f2$ where $f3 = BU_{i2}(f4)$, $f2 = View_{h2}(f4)$ and $f1 = View_{h1}(f3) = BU_{i1}(f2)$.

Proof

Since the diagram (a) is pushout in **BP** we have by the Behaviour Extension Lemma that $Free_{i2}+$ is an extension of $Free_{i1}$ which moreover is given by $Free_{i2}+(B) = B +_{B1} Free_{i1}(B1)$ where $B1 = BU_{h1}+(B)$. Thus in particular

$$Free_{i2}+(R_{h1(E1)}(A3)) = R_{h1(E1)}(A3) +_{A1} Free_{i1}(A1) = R_{h1(E1)}(A3) +_{A1} A2 = A4$$

The algebra A4 is a free construction w.r.t. A3 because by the View Extension Lemma $Free_{i2}$ is also an extension of $Free_{i2}+$.

The same argument is valid for morphisms.

4. Behaviour Constraints

As it was said in the introduction, our aim is to deal with incomplete behavioural specifications by means of constraints. The idea will be that a specification consists of a *global* presentation that includes all the sorts, operations and equations that have been declared up to a certain point and a set of constraints that characterize the completely defined subparts of the given specification. These constraints work as the standard free or generating constraints [BG 80, Rei 80] but only up to behavioural equivalence. That is, the use of standard free generating constraints allows to restrict the class of models of a given specification by considering acceptable only those models that satisfy that a certain subpart of the model has been freely constructed from another subpart. In our framework, making use of the existence of free constructions for categories of behaviours [Niv 87, NO 88] that work like the standard free constructions, but up to behavioural equivalence, we define our constraints by means of these behaviour free constructions. In this sense, intuitively, a model of a given specification satisfies a behaviour constraint if some part of this model is behaviourally equivalent to what can be obtained aplying a free construction to another subpart. It must be said that in fact, the situation is a little more complicated, as we will see later, because the *local* observability criteria used within the constraint need not to coincide with the global criteria used in the specification. However, before considering this problem let us, first, see an standard simple example of what we may consider an incomplete behavioural specification.

```
bspec Val_eq = enrich Bool with
                        obs sorts val
                        opns  eq: val val → bool
                        eqns  eq(x,x) = true
                              eq(x,y) = eq(y,x)
                              (eq(x,y) and eq(y,z)) ⇒ eq(x,z) = true
end bspec

bspec Set = enrich Val_eq defining
                        non-obs sorts set
                        opns  ∅: → set
                              add: set val → set
                              _∈_ : val set → bool
                        eqns  add(add(s,x),y) = add(add(s,y),x)
                              x∈ ∅ = false
                              x∈ add(s,y) = (x∈ s) or eq(x,y)
end bspec

bspec Choose = enrich Set with
                        opns  choose: set → val
                        eqns  choose(add(s,x))∈ add(s,x)
end bspec
```

According to our framework, the specification Choose is an incomplete specification with two completely defined subparts: the booleans and the sets of values. On the other hand, in Choose the sort val and the operations eq and choose are considered to be incompletely defined. The semantics of this specification is going to be loose, i.e. all (behavioural) models of the specification satisfying the constraints will be considered admissible. In particular, this means that admissible models will be those that their Boolean part coincides with the standard boolean algebra of two elements and whose Set part behaves as finite sets of elements taken from the sort val. Note that this means that, if the Set part are

sequences of values, this will be an admissible model, even if it is not a model in the standard sense (it does not satisfy the commutativity property for add).

Now, in order to define the *proper* notion of behaviour constraints we have to take into account that, as said above, observability in constraints must be *local* and not *global* in the following sense: In the standard framework that we defined in [OSC 89] the presentations defining the constraint on a given specification were contained in the *global presentation*. Here, asking for this inclusion would not be sensible, in general, because it could happen that none of the sorts involved in the constraint is observable and, as a consequence, the semantics of the constraint would be trivial. This means that some sorts should be considered locally observable within the constraint even if, at the global level, they are not observable. This also means that, in order to define constraint satisfaction, i.e. to describe how some parts of the models of the given specification are freely constructed (up to behavioural equivalence) from another part of the model, the forgetful functor cannot be used to *obtain* these parts. Instead, a View functor will have to be used.

Definition 4.1

A **behaviour constraint** BC is a pair of behaviour presentations (BP1, BP1') such that Obs1 = S1 and BP1 $\subseteq$ BP1'.

Given a presentation BP, a **behaviour constraint** BC = (BP1, BP1') is **defined on** BP if 1) BP1' is *view included* in BP, i.e. P1' $\subseteq$ P and S1'-Obs1' $\subseteq$ S-Obs, and 2) for every sort s1 in Obs1'-Obs1 we have that s1 is in Obs.

An algebra A $\in$ Beh(BP) **satisfies** a behaviour constraint (BP1, BP1') defined on BP, denoted A $\models$ (BP1, BP1') if

$$(A|_{BP1})|^{BP1'} \equiv_{BP1'} A|_{BP1'}$$

Notation:

We will shortly write $\equiv$ instead of $\equiv_{BP}$ if Bp is clear from the context. From now on, we will denote by $\_|_{BP}$: Beh(BP') $\rightarrow$ Beh(BP) and $\_|^{BP'}$: Beh(BP) $\rightarrow$ Beh(BP') the forgetful functor and the free functor respectively which are associated to the inclusion BP $\subseteq$ BP'. Moreover, if BP is view included in BP' then $\_|_{BP}$: Beh(BP') $\rightarrow$ Beh(BP) will denote the View functor associated to this view inclusion.

The previous definitions reflect the above discussion. In particular, condition 1) states our choice with respect to local observability within a constraint, i.e. we have considered that when stating a constraint (BP1, BP1') all sorts in BP1 could be used to observe the behaviour of the objects *created* by the constraint. On the other hand, condition 2) states that all sorts introduced by the constraint should be observable if and only if they are observable at the global level. The reason for this is that we are considering that constraints are the way of completely defining the sorts and operations introduced by them, i.e. the sorts and operations that are in BP1'-BP1. Therefore, if this is the complete definition of these sorts, their observability should also be defined by the constraint, i.e. the observability of

these sorts should be the same within the constraint and at the global level.

Now, we can define our concept of behaviour (incomplete) specification as a presentation, including all the sorts, operations and equations of interest at this point and a set of constraints defining the complete parts of this specification. The semantics of such specification is, obviously, loose.

Definition 4.2

A **behaviour specification** BSP is a pair <BP, ζ> where BP is a behaviour presentation and ζ is a set of behaviour constraints on BP. The **semantics** of a behaviour specification BSP is defined by the following class of models

$$Mod(BSP) = \{A \in Beh(BP) \mid A \models \zeta\}$$

As in [OSC 89] and other related approaches (e.g. [ST 87b]) no special notion of specification (internal) correctness is used apart of consistency, i.e. the class of models of a given specification should not be empty.

As said above the basic constructions needed for adequately defining the operations for building specifications are the Amalgamation and Extension Lemmas. The Amalgamation Lemma we present here is just an extension of the one in section 3., in the sense that it applies to specifications (with constraints) and not only presentations. On the other hand, the Extension Lemma is an especial case of the View Extension Lemma from section 3, just considering that the subcategories of algebras on which we build the extension are the ones defined by the constraints.

In what follows we will define these lemmas with respect to pushouts in which all morphisms are inclusions or view inclusions. The reasons for this restriction is, on the one hand, simplicity and, on the other, that with the exception of some constructions at the end of the paper, that need that two of the arrows of a pushout be what it is called a refinement morphism, we only need inclusions.

Pushouts of specifications will not be explicitly defined although they are what it is expected, i.e. the pushout of the global presentations and if, we are just dealing with inclusions, the union of the sets of constraints.

Lemma 4.3 (Behaviour Amalgamation Lemma with Constraints)

Let BSPi = (BPi, ζi), i=1..4, be two specifications such that BSP4 is the pushout of BSP2 and BSP3 with respect to BSP1. Then

$$Mod(BSP4) = Mod(BSP2) +_{Mod(BSP1)} Mod(BSP3)$$

Moreover, amalgamation has the following universal property: If $A4 = A2 +_{A1} A3$ (resp. $h4 = h2 +_{h1} h3$) then A4 is the unique algebra (resp. h4 is the unique homomorphism) satisfying $A4 \mid_{P2} = A2$ and $A4 \mid_{P3} = A3$. (resp. $h4 \mid_{P2} = h2$ and $h4 \mid_{P3} = h3$).

Proof

By making use of the Behaviour Amalgamation Lemma, it is only necessary to prove that if $A4 = A2 +_{A1} A3$, and Ai is in Mod(BSPi), i=1,3, then A4 satisfies every constraint in BSP4. Now, let (BP, BP') be a constraint in BSP4. This means that there there is an i (i=1,2 or 3) such that (BP, BP') is in BSPi. But then we have:

$$A4 |_{BP} |^{BP'} = A4 |_{BPi} |_{BP} |^{BP'} = Ai |_{BP} |^{BP'} = Ai |_{BP'} = A4 |_{BPi} |_{BP'} = A4 |_{BP'}$$

[]

In order to state the Extension Lemma we need, we will first define the notion of relative persistency that is adequate here.

Definition 4.4

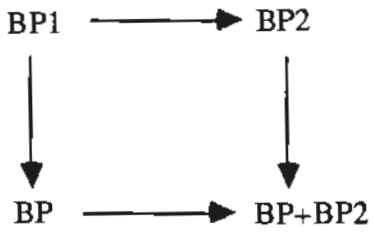
Given a specification BSP = <BP, ζ>, a behaviour constraint BC = (BP1, BP1') with BP1 ⊆ BP is **persistent relative to** BSP if for any A in Mod(BSP) it holds that

$$(A |_{BP1} |^{BP1'}) |_{BP1} = A |_{BP1}$$

Please note that we do not assume BP1' to be included in BP.

Lemma 4.5 (Extension Lemma with Constraints)

Let BSP be a specification, BSP = <BP, ζ>, and BC be a behaviour constraint, BC = (BP1, BP2), such that BP1 is view included in BP and BP∩BP2 = BP1 and let BP+BP2 denote the result specification of the pushout diagram



then if (BP1, BP2) is persistent relative to BSP we have that (BP, BP+BP2) is persistent relative to BSP and the associated free functor $\_ |^{BP+BP2}$: Beh(BP) → Beh(BP+BP2) is an extension of $\_ |^{BP2}$: Beh(BP1) → Beh(BP2) for BSP-models, that is for every A in Mod(BSP):

$$A |_{BP1} |^{BP2} = A |^{BP+BP2} |_{BP2}$$

Moreover, $Mod(BSP) |^{BP+BP2} \subseteq Mod(<BP+BP2, \zeta \cup \{(BP1, BP2)\}>)$ and for every algebra A4 (respectively behaviour homomorphism h4) in Mod(BSP4) we have that A4 (resp. h4) is naturally isomorphic to $A4 |_{BP} |^{BP+BP2}$ (respectively $h4 |_{BP} |^{BP+BP2}$).

Proof

Immediate from the View Extension Lemma and from the View Amalgamation Lemma for the free case just considering that the subcategories A1, A3 and A4 of Beh(BP1), Beh(BP3) and Beh(BP4) are the ones that satisfy the constraints of BSP . []

5. Main results

Now that we have the basic results needed (amalgamation and extension lemmas) we will extend the results presented in [OSC 89] to the behaviour case. These results concern the whole process of building a specification. In particular, first we will see that if a specification is completely defined then its semantics coincides with the initial behaviour semantics [Niv 87, NO 88], i.e. the final result of the development process has what we consider the proper behaviour semantics of a specification. Then, we will present the three basic operations for extending a specification (horizontal refinements) and show that we can define compatible semantics both at the model level and at the specification level. Finally, we introduce the notion of vertical refinement and show a horizontal composition theorem that may be seen as a generalization of parameter passing as defined in [EM 85] for the standard case and in [Niv 87, NO 88] for the behaviour case.

The notion of completeness of a specification is, as in [OSC 89] based on two properties 1) every sort and operation is defined in some constraint and 2) there is no circularity among constraints. The absence of circularity is needed as it shows the following example:

Let BP0, BP0', BP1, BP1' and BP be

```
bspec BP0 = obs sorts s1 end bspec

bspec BP0'= obs sorts s1, s2
            ops f: s1 → s2
end bspec

bspec BP1 =obs sorts s2 end bspec

bspec BP1'=  obs sorts s1, s2
             ops g: s2 → s1
end bspec

bspec BP =  obs sorts s1, s2
            ops f: s1 → s2
                g: s2 → s1
end bspec
```

and let BSP be (BP, {(BP0, BP0'), (BP1, BP1')}). In BSP every sort and operation seems to be defined on some constraint, but this is not really true. In fact, the constraints only say that elements of sorts s1 must be a copy of elements of sort s2 and vice-versa. Absence of this kind of circularity allows to avoid this kind of situations. Certainly, circularity is not by itself a problem (for instance, in the previous example another constraint could have existed really defining the elements of sorts s1 and s2). However, for simplicity, we have adopted this restricted notion together with the additional restriction that every sort or operation is defined by a unique constraint. Nevertheless the next theorem would also apply for not so strong restrictions.

Definition 5.1

A specification BSP = <BP, ζ> is **complete** if the following two conditions hold:

a) **Complete definition**: for every $s \in S$ there exists a unique $(BP1, BP2) \in \zeta$ such that $s \in S2\text{-}S1$ and for every $op \in \Omega$ there exists a unique $(BP1, BP2) \in \zeta$ such that $op \in \Omega 2\text{-}\Omega 1$.

b) **No circularity**: the transitive closure of the relation $<$, defined by $(BP1, BP2) < (BP3, BP4)$ if there exists $s \in S3$ such that $s \in S2\text{-}S1$ or there exists $op \in \Omega 3$ such that $op \in \Omega 2 - \Omega 1$, is a strict partial order on ζ.

Theorem 5.2

Let $BSP = \langle BP, \zeta \rangle$ be a consistent behaviour specification, then if BSP is complete we have

$$Mod(BSP) = \{A \in Alg(BP) \mid A \equiv_{BP} T_{BP}\}$$

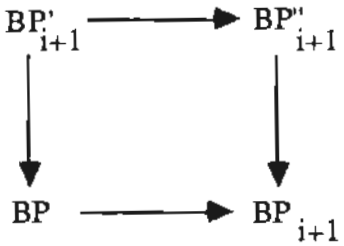
where T_{BP} is the initial BP-algebra.

Proof

Let $BC_0, BC_1, ..., BC_n$ be a topological sort of ζ with respect to the partial order defined by condition b. that is $BC_i < BC_j$ implies $i<j$. Note that BC_0 must have the form $(\emptyset, BP_0)$. Let $BSP_0 = \langle BP_0, \zeta_0 \rangle, ..., BSP_n = \langle BP_n, \zeta_n \rangle$ be the following sequence of specifications:

$$BSP_0 = \langle BP_0, (\emptyset, BP_0) \rangle$$
$$BSP_{i+1} = \langle BP_{i+1}, \zeta_i \cup \{ (BP'_{i+1}, BP''_{i+1}) \} \rangle$$

where $BC_{i+1} = (BP'_{i+1}, BP''_{i+1})$ and BP_{i+1} denotes the result of the pushout



Note that for every i, j, with $i \leq j$, if s is in $S''_i\text{-}S'_i$ then s is observable in BP''_i iff s is observable in BP_j. Now, we will prove by induction that for every i:

1. BC_i is persistent relative to BSP_{i-1}. In the case $i = 0$ we consider BC_i to be persistent (persistent relative to the empty specification), which trivially is since $BC_0 = (\emptyset, BP_0)$.

2. $T_{BPi} \in Mod(BSP_i)$

3. If $A, B \in Mod(BSP_i)$ then $A \equiv_{BSP_i} B$.

It should be clear that, if 1., 2. and 3. hold for every i, then the theorem is true since, by construction, $BSP_n \subseteq BSP$ and in addition, by condition a., $\Sigma_n = \Sigma$ and $\zeta_n = \zeta$. Then, $Mod(BSP) \subseteq Mod(BSP_n)$. But if $Mod(BSP_n)$ only contains algebras which are isomorphic to T_{BPn}

and Mod(BSP) cannot be empty, since it is assumed to be consistent, then Mod(BSP) = Mod(BSP_n) and $T_{BP_n} \equiv_{BP} T_{BP}$.

If i = 0 then, as it was said above, condition 1. trivially holds. Also, conditions 2. and 3. are obviously satisfied since the only BP_0-algebras that satisfy the behaviour constraint $(\varnothing, BP_0)$ are exactly the algebras which are isomorphic to T_{BP_0}.

Assume i = j+1. To prove that BC_{j+1} is persistent relative to BSP_j we have to prove that:

$$T_{BP_j} \mid BP'_{j+1} \mid^{BP''_{j+1}} \mid BP'_{j+1} \equiv T_{BP_j} \mid BP'_{j+1}$$

Now, if BSP is consistent there should be an A such that $A \in Mod(BSP)$, but since T_{BP_j} is the only BP_j-algebra satisfyimg the behaviour constraints in ζ_j, this means that $A \mid BP_j \equiv T_{BP_j}$. On the other hand, A must also satisfy the behaviour constraint BC_{j+1}, therefore:

$$A \mid BP'_{j+1} \mid^{BP''_{j+1}} \equiv A \mid BP''_{j+1}$$

but this implies that:

$$T_{BP_j} \mid BP'_{j+1} \mid^{BP''_{j+1}} \equiv A \mid BP_j \mid BP'_{j+1} \mid^{BP''_{j+1}} \equiv A \mid BP''_{j+1}$$

and therefore:

$$T_{BP_j} \mid BP'_{j+1} \mid^{BP''_{j+1}} \mid BP'_{j+1} \equiv A \mid BP''_{j+1} \mid BP'_{j+1} = A \mid BP'_{j+1} \equiv T_{BP_j} \mid BP'_{j+1}$$

Now, to prove 2. it is enough to notice that, since (BP'_{j+1}, BP''_{j+1}) is persistent relative to BSP_j, according to the Extension Lemma $T_{BP_j} \mid^{BP_{j+1}}$ is in $Mod(BSP_{j+1})$. But, $T_{BP_j} \mid^{BP_{j+1}} \equiv T_{BP_{j+1}}$.

Finally, 3 is also a consequence of the Extension Lemma. On one hand we have that all algebras in $Mod(BSP_j)$ are isomorphic which implies that all algebras in $Mod(BSP_j) \mid BP'_{j+1}$ are also isomorphic and, therefore, so it happens with algebras in $Mod(BSP_j) \mid BP'_{j+1} \mid^{BP''_{j+1}}$. On the other, from the Extension Lemma we have that:

$$Mod(\langle BP_{j+1}, \zeta_j \cup \{BC_{j+1}\}\rangle) = Mod(BSP_j) \oplus_{Beh(BP'_{j+1})} (Beh(BP'_{j+1}) \mid^{BP''_{j+1}})$$

then, from the Amalgamation Lemma [EM85], we have that all algebras in $Mod(\langle BP_{j+1}, \zeta_j \cup \{BC_{j+1}\}\rangle)$ are also isomorphic. []

The next thing to study, as said above, are the basic operations that we define for building a specification. We consider two kinds of them: extension operations and completion operations. Extension operations, which correspond to horizontal refinements, build larger specifications on top of existing ones in a conservative way. That is, we assume that if BSP2 extends BSP1 then the models of BSP2, when forgetting the new sorts and operations coincide exactly with the models of BSP1. This means that we assume that extension operations do not add additional detail on existing sorts and operations, i.e. there is no restriction on the class of models. On the other hand, completion operations, which correspond to vertical refinements, do restrict the class of models of the refined specification. Since completion operations may also add new sorts, operations and equations to a existing

specification, it happens that extensions are a special case of vetical refinements. Then, we could ask about the need of this distinction. The reason is mainly methodological, we believe that a specifier should always be conscious of when is s/he adding new things or when is s/he adding detail or completing a previously existing specification. This also happens in many specification languages (e.g. the *protecting* case for enrichment declaration in OBJ [FGJ 85]). In particular, the language GSBL developped following the ideas introduced in [OSC 89] makes heavy use of this distinction in order to enhance the incremental construction of specificataions. Moreover, at a more technical level, knowing that some specifications are extensions, in our sense, of some subspecifications allows to assure, for free the correctness (i.e. consistency) of the result specification after applying certain operations. For instance this happens when *combining* specifications or when doing a horizontal composition of vertical refinements.

Definition 5.3
Given specifications BSP1 and BSP2, BSP2 is a **loose extension** of BSP1 if

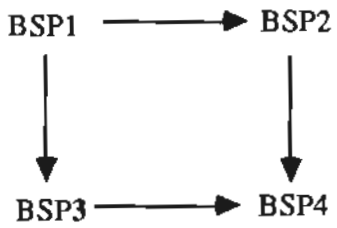
a) BSP1 $\subseteq$ BSP2
b) $Mod(BSP1) = Mod(BSP2)|_{BP1}$

We consider three basic operations for defining loose extensions: **enrich defining, enrich with** and **combine**. Their semantics, at the specification level is given below.

The operation **enrich defining** adds to a given specification new sorts and operations together with a constraint defining them. That is, given a specification BSP = <BP, ζ> and a constraint C = (BP1,BP2) such that BP1 $\subseteq$ BP, **enrich defining** creates a new specification <BP+BP2, $\zeta \cup \{C\}$>, where BP+BP2 denotes, as in the Extension Lemma (cf. 4.5) the pushout of BP and BP2 over BP1.

The operation **enrich with** adds new sorts and operations without any new constraint. That is, given a specification BSP = <BP, ζ>, where BP = ((S, Op), E), and a triple (S1, Op1, E1), such that BP1 = ((S+S1, Op+Op1), E+E1) is a presentation and where + denotes disjoint union, **enrich with** creates the new specification <P1, ζ>.

Finally, the operation **combine** puts together two specifications whithout duplicating their common part. That is, given specifications BSP1, BSP2 and BSP3, such that BSP2 and BSP3 are loose extensions of BSP1, the combination of BSP2 and BSP3 over BSP1 is defined as the result of the pushout:



The semantics of these three operations could also be defined at the model level as follows:

$$Mod(\textbf{enrich } <BP, \zeta> \textbf{ defining } (BP1,BP2)) = \{A \mid \exists\, A' \in Mod(<BP, \zeta>),\ A \equiv A'|^{P+P2}\}$$

$$Mod(\textbf{enrich } <BP, \zeta> \textbf{ with } (B\Sigma,E)\ ) = Mod(<BP, \zeta>) +_{Mod(<BP, \varnothing>)} Mod(<BP+(B\Sigma,E), \varnothing>)$$

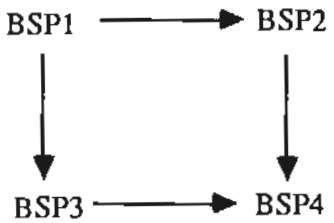
$$Mod(\textbf{combine } BSP2 \textbf{ and } BSP3 \textbf{ wrt } BSP1) = Mod(BSP2) +_{Mod(BSP1)} Mod(BSP3)$$

This model level definitions are compatible with the previous ones because of the Amalgamation and Extension Lemmas seen in the previous section. It may be noted that these definitions are essentially the same to the ones given in [OSC 89] except for the case of the **enrich defining** operation. The reason is that in [OSC 89] this operation was defined in terms of amalgamated sums, while here, because this operation involves some view inclusions, would not be possible, in general. As a consequence, in this paper, we have defined the meaning of that operation by means of the closure, under behavioural equivalence (behavioural isomorphy) of the extensions of the models of the enriched specification.

In what follows, we will study the correctness of these three operations, i.e. under which conditions these operations define loose extensions. The simplest case is the **combine** operation, since the result BSP4 of the combination of two specifications, BSP2 and BSP3, that are loose extensions of BSP1 is always a loose extension of BSP2 and BSP3:

Theorem 5.4

Let BSP1, BSP2 and BSP3 be three consistent specifications such that BSP2 and BSP3 are loose extensions of BSP1 and BSP2∩BSP3 = BSP1, and let BSP4 be the result of the pushout:



then BSP4 is a loose extension of BSP2 and BSP3.

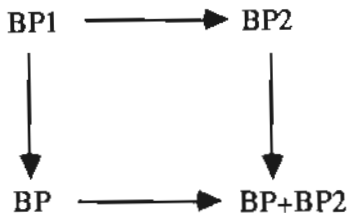
Proof

The proof is almost trivial: w.l.o.g., let us prove that BSP4 is a loose extension of BSP2. Let A2 be in Mod(BSP2), then we know that $A1 = A2|_{BP1}$ is in Mod(BSP1) and, since BSP3 is an extension of BSP1, there should be an A3 in Mod(BSP3) such that $A1 = A3|_{BP1}$. Then, by the Amalgamation Lemma for specifications with constraints [Ehg88], we have that $A4 = A2 +_{A1} A3$ is in Mod(BSP4). []

The case of **enrich defining** is also quite simple. It depends on the relative persistency of the new constraint with respect to the enriched specification:

Theorem 5.5

Given a specification BSP = <BP, ζ> and a constraint C = (BP1, BP2) such that $BP1 \subseteq BP$ and BP2∩BP = BP1, and let BP+BP2 be the result of the pushout:



then BSP' = <BP+BP2, ζ∪{C}> is a loose extension of BSP iff C is persistent relative to BSP.

Proof

If BSP' is a loose extension of BSP this means that for every BP-algebra A such that A |= ζ there is a BP+BP2-algebra B such that B |= ζ∪{C} and $B|_{BP} = A$. Now, if B satisfies C this means that:

$$B|_{BP1}|^{BP2} = B|_{BP2}$$

but this implies that:

$$B|_{BP1}|^{BP2}|_{BP1} = B|_{BP2}|_{BP1} = B|_{BP1}$$

and therefore:

$$A|_{BP1}|^{BP2}|_{BP1} = B|_{BP1}|^{BP2}|_{BP1} = B|_{BP1} = A|_{BP1}$$

Conversely, if (BP1,BP2) is persistent relative to BSP, by the Extension Lemma proved above, we know that for every A∈Mod(BSP) it holds that $A|^{BP+BP2} \in$ Mod(BSP') and $A|^{BP+BP2}|_{BP} = A$. []

Finally, the correctness of the **enrich with** operation is the most complicated case. Here, as in [OSC 89] we will just give a sufficient condition which we think can handle many situations. Essentially, it says that an enrichment of this kind over a specification BSP is a loose extension if we can provide a constraint persistent relatively to BSP, *defining completely* the enrichment. We think that this is a reasonable condition for many situations since, often, the reason of adding some sorts or operations without defining them completely is that we do not want to take a decision of choosing among several possible alternatives.

Corollary 5.6

Given a specification BSP = <BP, ζ>, and a presentation BP1, such that BP ⊆ BP1, then BSP1 = <BP1, ζ> is a loose extension of BSP if there exists a constraint C = (BP2, BP3), such that BP2 ⊆ BP, BP1 ⊆ BP3 and (BP2, BP3) is persistent relative to BSP.

Proof

If there is a constraint (BP2, BP3) such that BP2 ⊆ BP, BP1 ⊆ BP3 and (BP2, BP3) is persistent relative to BSP, then using the previous theorem we know that BSP' = <BP+BP3, ζ∪{C}> is a loose extension of BSP. But, $Mod(BSP')|_{BP1} \subseteq$ Mod(BSP1) thus BSP1 is a loose extension of BSP.[]

The second kind of refinements we consider are vertical refinements. A vertical refinement consists on *adding detail* to a specification, in our case completing the given specification or, similarly, restricting its class of models. In this sense, it seems reasonable to consider vertical refinements as some class of specification morphism. As in [OSC 89] we have considered a definition which is more restrictive than it, perhaps, could be. In particular, we have restricted *refinement morphisms* to translate constraints injectively. The reason for this is, mainly, methodological. According to our approach a constraint represents a part of a specification completely defined. In this sense, it seems reasonable to think that when we are completing a specification the already completed parts should remain *untouched*. A similar restriction is taken in [ETLZ82] but, apparently, just for technical reasons.

Definition 5.7

A **refinement morphism** f: <BP1, $\zeta 1$> $\rightarrow$ <BP2, $\zeta 2$> is a behaviour presentation mophism f: BP1 $\rightarrow$ BP2, satisfying:

a) f is injective on constrained sorts and operations, that is for every constraint (BP, BP') in $\zeta 1$, if s1, s2$\in$ S'-S (resp. op1, op2$\in \Omega' - \Omega$) then f(s1) = f(s2) implies s1 = s2 (resp. f(op1) = f(op2) implies op1 = op2)

b) $f(\zeta 1) \subseteq \zeta 2$.

Facts 5.8

1. Obviously, the composition of vertical refinements is a vertical refinement. Therefore, vertical composition trivially holds.

2. If f: <BP1, $\zeta 1$> $\rightarrow$ <BP2, $\zeta 2$> is a refinement morphism then $Mod(<BP2,\zeta 2>)|_{BP1} \subseteq Mod(<BP1, \zeta 1>)$. This is a consequence of the restriction imposing f to be injective on $\zeta 1$.

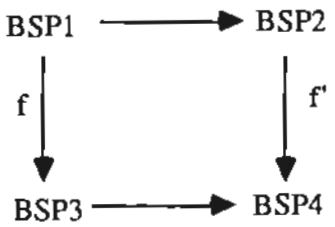
3. There are pushouts (amalgamations) associated to categories of specifications (models) with refinement morphisms (the associated forgetful functors). In particular, the proof of existence of amalgamations, in this case, would be just a slight generalization of the one given in the previous section.

The main operation for defining vertical refinements is presented in the following theorem. In particular, it shows how we can substitute, within a specification, an incomplete part for a more complete one. Specifically, it states how a vertical refinement of a given specification BSP1 induces a vertical refinement on any loose extension of BSP1. This fact has several interpretations. On one hand, the theorem states, in our framework, the horizontal composition property [GB80], namely, that the order in which we perform vertical and horizontal refinements is not important. On the other, it shows that in our framework there is no need for parameterization, since any specification BSP2 may be seen as having as implicit parameters all specifications BSP1 loosely extended by BSP2. Then, this induced vertical refinement may be seen as a generalized form of parameter passing. The relation of our construction to parameter passing is very similar to the one found by B. Meyer [Mey86], at the programming language level, between genericity and inheritance, showing that inheritance may be seen as a generalization of genericity. Indeed, as it is shown in [CO88], our notion of vertical refinement may be seen, from a methodological standpoint, as an inheritance relation defined at the specification level. Obviously, this kind of inheritance relation has nothing to do with the subtyping (or subsorting)

relation also studied in the literature [GM83].

Theorem 5.9

Let BSP1 and BSP2 be consistent behaviour specifications such that is a loose extension of BSP1, and let f be a refinement morphism, f: BSP1 → BSP3, for a given specification BSP3 such that BSP1 = BSP2∩BSP3. The result of substituting BSP1 by BSP3 in BSP2 is the specification BSP4 = <BP4,ζ4> defined by the pushout:



then we have:

1. BSP4 is a loose extension of BSP3
2. BSP3 is consistent iff BSP4 is consistent

Proof

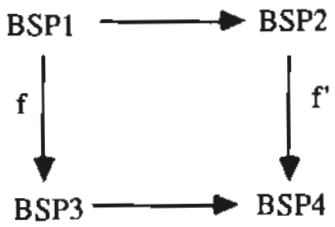
1.We know that Mod(BSP4) = Mod(BSP3) $+_{Mod(BSP1)}$ Mod(BSP2). Then, if Mod(BSP3) = ∅, so is Mod(BSP4). Now, given an A∈ Mod(BSP3), we have that $A|_{BP1} \in$ Mod(BSP1). But, if BSP2 is a loose extension of BSP1, there is a B∈ Mod(BSP2) such that $B|_{BP1} = A|_{BP1}$. Then, by defining $B' = A +_{A|_{BP1}} B$ we have that B'∈ Mod(BSP4) and $B'|_{BP3} = A$.

2. Is an immediate consequence of 1., since if BSP4 is a loose extension of BSP3, then Mod(BSP3) is not empty iff Mod(BSP4) is not empty.[]

In the previous theorem, the fact that BSP2 is a loose extension of BSP1, i.e. $Mod(BSP2)|_{BP1}$ = Mod(BSP1), is absolutely needed to guarantee the consistency of BSP4. The situation is similar to the need of persistency to assure the correctness of parameter passing:

Theorem 5.10

Let BSP1 and BSP2 be specifications such that BSP1 ⊆ BSP2, then if BSP2 is not a loose extension of BSP1 there is a consistent specification BSP3 such that BSP2∩BSP3 = BSP1 and a refinement morphism f: BSP1 → BSP3 such that the result, BSP4, of the associated pushout diagram:



is not consistent.

Proof

If BSP2 is not a loose extension of BSP1 this means that there is an A1∈ Mod(BSP1) such that

for every A2∈Mod(BSP2) $A2|_{BP1}$ is not isomorphic to A1. Let BSP3 = <BP3, ζ3>, where BP3 is the presentation obtained by adding to BP1 all the values from A1 as constants of the appropriate sorts and all the equations satisfied by A1, and ζ3 is obtained by adding to ζ1 the constraint (∅, BP3). Obviously, Mod(BSP3) = $\{B \in Beh(BP3) / B|_{BP1} \cong A1\}$.
Now, Mod(BSP4) = ∅, for

$$Mod(BSP4) = Mod(BSP3) +_{Mod(BSP1)} Mod(BSP2)$$

and

$$\{A \in Mod(BSP2) / A|_{BP1} \in Mod(BSP3)|_{BP1}\} = \emptyset \quad []$$

6. Conclusions

We have presented an approach for formalizing the specification development process from informal requirements which is capable of dealing with incomplete specifications. Moreover, it also takes into account the notions of behaviour and observability which are critical with respect to the semantics of software specifications. The way of handling this incompleteness is by means of behavioural specifications with constraints which correspond to the completely defined subparts of a given incomplete specification. The concept of behavioural constraint allows to deal with different observability criteria within the same behavioural specification (certain sorts are considered non-observable at the global level but may be considered locally observable within a constraint) which is a need from the metodological point of view. The corresponding semantics is loose, accepting as models all algebras behaviourally satisfying the axioms and all the behavioural constraints, obtaining compatibility results with respect to the operations for horizontal and vertical refinement. The *interaction* of horizontal and vertical refinements allows to substitute any incomplete subspecification of a given specification by a more complete one in a way that generalizes the satandard behavioural parameter passing by requiring a limited form of persistency in the behaviour sense.

To achieve that, a generalization of the Amalgamation and Extension Lemmas for behaviour specification morphisms, the Extension Lemma for view morphisms and a version of the Amalgamation for view morphisms for free algebras has been necessary to obtain.

7. References

[BG 80] R.M. Burstall, ; J.A. Goguen, The semantics of Clear, a specification language, Proc. Copenhagen Winter School on Abstract Software Specification, Springer LNCS 86, pp. 292-332, 1980.

[CO 88] S. Clerici, ; F. Orejas, GSBL: an algebraic specification language based on inheritance, Proc. Europ. Conf. on Object Oriented Programming, (Oslo, 1988), Springer LNCS.

[Ehg 89] H. Ehrig, A categorical concept of constraints for algebraic specifications, Proc. International Workshop on Categorical Methods in Computer Science with Aspects from Topology, Berlin 1988, Springer LNCS 332, (1989).

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

[EPO 89] H. Ehrig, P. Pepper, F. Orejas, On recent trends in algebraic specification, Proc. ICALP 89, LNCS 372 (1989) 263 - 289.

[ETLZ 82] H. Ehrig, H., J.W. Thatcher, P. Lucas, S.N. Zilles, Denotational and initial algebra semantics of the algebraic specification language LOOK, Draft Report, IBM Research, 1982.

[FGJ 85] K. Futatsugi, J.A. Goguen, J.P. Jouannaud, Principles of OBJ 2, Proc. POPL 85, ACM (1985) 221-231.

[GB 80] J.A. Goguen, R.M. Burstall, CAT, a system for the structured elaboration of correct programs from structured specifications, Tech. Report CSL-118, Comp. Sc. Lab., SRI Int., 1980.

[GB 85] J.A. Goguen, R.M. Burstall, Institutions: Abstract Model Theory for Computer Science, CSLI Report 85-30, 1985.

[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.

[Mey 86] B. Meyer, Genericity versus Inheritance, Proc. ACM conf. Object-Oriented Programming Syst, Languages, and Applications, ACM, New York, 1986, pp. 391-405

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

[NO 88] Mª P. Nivela, F. Orejas, Initial Behaviour Semantics for Algebraic Specifications, Proc. 5th Workshop on Algebraic Specifications of Abstract Data Types, Gullane 1987, Springer LNCS 332, (1988) 184-207.

[ONE 89] F. Orejas, Mª P. Nivela, H. Ehrig, Semantical constructions for categories of behavioural specifications, Proc. International Workshop on Categorical Methods in Computer Science with Aspects from Topology, Berlin 1988, Springer LNCS 332, (1989) 220-243.

[OSC 89] F. Orejas, V. Sacristan, S. Clerici, Development of Algebraic Specifications with Constraints, Proc. International Workshop on Categorical Methods in Computer Science with Aspects from Topology, Berlin 1988, Springer LNCS 332, (1989)

[Rei 80] H. Reichel, Initially restricting algebraic theories, Proc. MFCS 80, Springer LNCS 88 (1980), pp. 504-514.

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

[Rei 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.

[Rei 87] H. Reichel, Initial Computability, Algebraic Specifications and Partial Algebras, Int. Series of Monographs on Comp. Sc, Oxford Science Publ., 1987.

[SW 83] 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.

[ST 87a] D. Sannella, A. Tarlecki, On observational equivalence and algebraic specification. J. Comp. and Sys. Sciences 34, pp. 150-178 (1987).

[ST 87b] D. Sannella, A. Tarlecki Toward formal development of programs from algebraic specifications: implementations revisited. Proc. Joint Conf. on Theory and Practice of Software Development, Pisa, Springer LNCS 249, pp. 96-110 (1987).