REPORTS
IN
INFORMATICS

ISSN 0333-3590

Specification of Parameterized Data Types

Yngve Lamo and Michał Walicki

REPORT NO 210 **December 2000**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL
`http://www.ii.uib.no/publikasjoner/texrap/ps/2000-210.ps`

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at `http://www.ii.uib.no/publikasjoner/texrap/`.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret, P.O. Box 7800, N-5020 Bergen, Norway

Specification of Parameterized Data Types

Yngve Lamo*
Department of Engineering
Bergen College
N-5020 Bergen
Norway

Michał Walicki†
Department of Informatics
University of Bergen
N-5020 Bergen
Norway

27th December 2000

1 Introduction

The important distinction between parameterized specifications and specifications of parameterized programs has been originally pointed out in [11]. An important difference concerns the objects which the two kinds of parameterization allow one to reuse.

A *parameterized specification*, "PSP", e.g., of stacks parameterized by elements, **Stack(El)**, allows one, under some conditions, to instantiate the formal parameter specification **El** with an actual parameter specification, say **Nat**, to obtain a new specification **Stack(Nat)**. What is reused is the *text* of the parameterized specification. Except for that, the model classes of **Stack(El)** and **Stack(Nat)** are just model classes of two different specifications, related merely by the existence of a reduct functor from the latter to the former, e.g. [2].

Specification of a parameterized data type, "PDT" [12, 11], on the other hand, is a specification which requires a reusable *implementation*. E.g., specification of a data type stack parameterized by elements, **Stack[El]** requires an *implementation of a data type with a parameter*, i.e., one capable of taking any implementation of **El** and resulting in an implementation of **Stack[El]**. The model class of such a specification should be thus seen as consisting of some – perhaps all – functors sending models of the formal parameter specification **X** to models of the parameterized specification **P**[**X**] :

$$\mathsf{FMod}(\mathbf{P}[\mathbf{X}]) \subseteq \{F : \mathsf{Mod}(\mathbf{X}) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}])\}. \quad (1)$$

There has been surprisingly little work done wrt. to this latter kind of parameterization (recent work on CASL, [6], is a valuable exception). "Surprisingly" because the idea of specifying actual architecture of designed software should appeal to one's sense of potential applicability of specifications. We mention a few possible reasons of this negligence which also indicate the approach of this paper.

Study of PSPs has long tended in the direction of PDTs [1, 2, 3, 5]. One of the problems is that, while the former continued the tradition of working with classes axiomatized by (possibly conditional) equations, the latter require a precise grasp on individual algebras (which, for modeling purposes, can be identified with programs): a program P taking as a parameter another program X cannot change X – X functions in the context of P, that is in $P[X]$, in the same way as it would in isolation. This intuition of "preserving actual parameter" has been identified as one of the semantic requirements on PSP in form of the persistency requirement on the functors from (1), e.g., [3, 14, 2]. However, in the purely equational context, there was hardly any syntactic

*Email: yngvel@ii.uib.no
†Email: michal@ii.uib.no

counterpart of this semantic requirement. Thus, no syntactic/logical means were available for reasoning about correctness of such implementations.

Even worse, persistency turns out to be all too strong a requirement eliminating many (if not most) interesting examples of PDTs as "illegal". For instance, a functor which takes an **El**-algebra of elements and adds a new element (intended, e.g., as the "error value" resulting from inspecting the top of an empty stack in the specification **Stack[El]**) is *not* persistent. In general, most free functors are not persistent, since these involve, typically, generation of new elements. Partial algebras admit more free functors than the total ones (since "error" elements remain simply undefined), but they still exclude the possibility of adding "new" elements and, in particular (as pointed out e.g. in [7]), of explicit error treatment.

We introduce a more adequate framework for specifying PDTs. The first thing is a generalization of the classical concept of persistent functors, so that our semantic functors can add new elements to the parameter algebras. This is the basic idea (going back to the short paper [10]) underlying our approach.

Although the semantic definition of the desirable functors is simple, a lot of book-keeping is required on the side of the syntax. Section 3 begins by introducing the syntactic preliminaries needed for specification of PDTs, and then defines the syntax and semantics of PDTs. Section 4 discusses syntax and semantics of actual parameter passing. Section 5 shows the counterparts of the classical, vertical and horizontal composition theorems. In this connection, we also encounter the concept of refinement of PDTs. Since PDTs correspond more to the design-, and not only to the requirement-specifications, their refinement reflects more specific design decisions. Unlike the classical concept of model class inclusion, refinement of PDTs amounts to introduction of additional structure. For instance, identifying a part of a (flat) specification as a parameter, amounts to requiring a structured, i.e., parameterized (rather than a flat) implementation. This section leads to a general concept of such a refinement, exemplifying the idea of "constructor specifications" from [12], which is summarized in section 6. Section 7 contains some concluding remarks.

The framework is presented using multialgebras [16, 17] and thus, in particular, allowing unrestricted use of nondeterministic data types. However, we do not focus on the issue of nondeterminism here and multialgebras are chosen merely as an example of a framework capable of expressing the intended concepts. It should be emphasised that entirely analogous definitions and constructions to those presented here can be done in any institution where the category of signatures has pushouts, the model functor is (finitely) continuous (or else, with amalgamation property) and where the signatures can express predicates. Thus our results can be used in a straightforward way, for instance, for membership algebras [9], for total/partial algebras with predicates, etc. This is one of the reasons why we did not attempt to define a new institution, but introduced the required concepts "on the top" of the institution of multialgebras.

The reader is assumed to have elementary knowledge of category theory and algebraic specifications. Some familiarity with the concept of institution would be advantageous but isn't necessary. To make the paper (almost) self-contained, we begin now, in section 2, by reviewing the relevant concepts from institutions and, in particular, the institution of multialgebras.

2 Preliminaries – institution of multialgebras

We list here only relevant definitions and facts. For more detailed discussion of these concepts and proofs of the theorems concerning institutions, the reader is referred, for instance, to [13], while for those concerning the institution of multialgebras to [8]. We start with the general concepts related to institutions. The reader completely unfamiliar with institutions may go directly to subsection 2.2 and inspect subsection 2.1 only if necessary later on.

2.1 Institutions

An institution [4, 13] is an abstract form for a logical framework. Showing that a particular specification formalism forms an institution allows one to apply some facts, properties and constructions which have been demonstrated generally for all, or some particular institutions.

Definition 2.1 *An institution is a quadruple $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$, where:*

- $\mathbf{Sign}$ *is a category of signatures.*
- $\mathsf{Sen} : \mathbf{Sign} \to \mathbf{Set}$ *is a functor which associates a set of sentences to each signature.*
- $\mathsf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *is a functor which associates a category of models, whose morphisms are called Σ-morphisms, to each signature Σ*
- $\models$ *is a satisfaction relation – for each signature Σ, a relation $\models_\Sigma \subseteq |\mathsf{Mod}(\Sigma)| \times \mathsf{Sen}(\Sigma)$, such that the following satisfaction condition holds: for any $M' \in \mathsf{Mod}(\Sigma'), \mu : \Sigma \to \Sigma', \phi \in \mathsf{Sen}(\Sigma)$*
$$M' \models_{\Sigma'} \mathsf{Sen}(\mu)(\phi) \quad \textit{iff} \quad \mathsf{Mod}(\mu)(M') \models_\Sigma \phi$$

Given an institution, one defines the category of theories (or specifications), $\mathbf{Th}$, with objects being pairs (Σ, Γ) of a signature Σ and a set of Σ-sentences Γ. Models of a theory are obtained by restricting the Mod functor, i.e., $\mathsf{Mod}(\Sigma, \Gamma) = \{M \in \mathsf{Mod}(\Sigma) : M \models_\Sigma \Gamma\}$. A morphism in $\mathbf{Th}$, $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$, is a *specification morphism*, that is, a signature morphism $\mu : \Sigma \to \Sigma'$ such that $\Gamma' \models \mu(\Gamma)$.

Putting theories together is done by constructing their co-limits. The following institution independent result ensures that the category of specifications has all co-limits if the signature category has. This result is used to create co-limits of specifications by first creating the co-limit for the corresponding signatures. The functor Sign forgets the axioms and retains the signature of a specification.

Proposition 2.2 *The functor* $\mathsf{Sign} : \mathbf{Th} \to \mathbf{Sign}$ *reflects co-limits, in any institution $\mathcal{I}$.*

In particular, passing an actual parameter specification to a parameterized specification corresponds to constructing a pushout. The desirable property is that such a construction at the level of syntax be reflected at the level of model classes. This is ensured in a particular kind of institutions, which will be of importance in this paper, the *exact* institutions.

Definition 2.3 *An institution $\mathcal{I}$ is*

1. semi-exact *iff* $\mathbf{Sign}$ *has pushouts and* Mod *sends pushouts in* $\mathbf{Sign}$ *to pullbacks in* $\mathbf{Cat}$*,*
2. exact *iff* $\mathbf{Sign}$ *has finite co-limits and* Mod *sends finite co-limits in* $\mathbf{Sign}$ *to limits in* $\mathbf{Cat}$*.*

Although the weaker property of semi-exactness is sufficient for treatment of parameterization and actualization, we will work with an exact institution.

Semi-exactness is of uttermost relevance for treatment of PDTs because it implies the following, more specific property.

Lemma 2.4 *(Amalgamation Lemma).*
In any semi-exact institution $\mathcal{I}$, for every pushout of signatures (on the left):

$$\begin{tikzcd}
\Sigma \arrow[r, "\mu_1"] \arrow[d, "\mu_2"'] & \Sigma_1 \arrow[d, "\mu'_2"] & & \mathsf{Mod}(\Sigma) & \mathsf{Mod}(\Sigma_1) \arrow[l, "-|_{\mu_1}"'] \\
\Sigma_2 \arrow[r, "\mu'_1"'] & \Sigma' \arrow[r, phantom, "\overset{\mathsf{Mod}}{\Longrightarrow}"] & & \mathsf{Mod}(\Sigma_2) \arrow[u, "-|_{\mu_2}"] & \mathsf{Mod}(\Sigma') \arrow[l, "-|_{\mu'_1}"] \arrow[u, "-|_{\mu'_2}"']
\end{tikzcd}$$

we have that: for any two models $M_1 \in \mathsf{Mod}(\Sigma_1)$ and $M_2 \in \mathsf{Mod}(\Sigma_2)$ satisfying $M_1|_{\mu_1} = M_2|_{\mu_2}$, there exists a unique model $M' \in \mathsf{Mod}(\Sigma')$, such that $M'|_{\mu'_1} = M_2$ and $M'|_{\mu'_2} = M_1$.

(The amalgamation lemma holds then also for pushouts of specifications, since these are constructed from pushouts of signatures by proposition 2.2.) The amalgamation lemma tells that the model class of a pushout Σ' of signatures along μ_1, μ_2 is a pullback (in **Cat**) of the respective morphisms $\_|_{\mu_1}$, $\_|_{\mu_2}$. The model M', denoted $M_1 \oplus_M M_2$ (where $M = M_1|_{\mu_1} = M_2|_{\mu_2}$), is called the *amalgamated sum* of M_1 and M_2. This property allows one to give a uniform definition of functorial semantics of actual parameter passing, as we will see in a later section.

2.2 Multialgebras

Signatures for multialgebras are the standard algebraic signatures, i.e., pairs $(\mathbf{S}, \Omega)$ where $\mathbf{S}$ is a set of sort names, and Ω a set of operation names with argument sorts and result sort from $\mathbf{S}$. Also terms (over a signature Σ, with variables X, denoted $T(\Sigma, X)$) are built in the usual way. Unlike in standard algebras, operation in a multialgebra may return (possibly empty) sets of elements.

Definition 2.5 *A* multialgebra *A for a signature $\Sigma = (\mathbf{S}, \Omega)$ is given by:*

- *a set s^A, the carrier set, for each sort symbol $s \in \mathbf{S}$*
- *a subset $c^A \in \mathcal{P}(s^A)$, for each constant, $c :\to s$*
- *an additive operation $\omega^A : s_1^A \times \cdots \times s_k^A \to \mathcal{P}(s^A)$ for each symbol $\omega : s_1 \times \cdots \times s_k \to s \in \Omega$*

"Additivity" means that operations are composed pointwise by collecting the results of applications to all elements of argument sets, i.e., for sets of elements $t_1, \ldots, t_n$ (of appropriate sorts) and operation $\omega : \omega^A(t_1, \ldots, t_n)) = \bigcup_{a_i \in t_i} \omega^A(a_1, \ldots, a_n)$.

Note that for a constant $c \in \Omega$, c^A denotes possibly a subset of the carrier s^A. This allows one to use constants as predicates as was done, for instance, in [7]. This will be the main aspect of multialgebras used in this paper.

As homomorphisms of multialgebras, we use weak homomorphisms (see [15] for alternatives).

Definition 2.6 *Given two multialgebras A and B, a function $h : |A| \to |B|$ is a (weak) homomorphism if:*

1. *$h(c^A) \subseteq c^B$, for each constant $c :\to s$*
2. *$h(\omega^A(a_1, \ldots, a_n)) \subseteq \omega^B(h(a_1), \ldots, h(a_n))$, for each operation $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$ and for all $a_i \in s_i^A$.*

Saying "homomorphism" we will always mean weak homomorphism. The collection of all Σ-multialgebras with Σ-homomorphisms forms a category.

Multialgebraic specifications are written using the following formulae:

Definition 2.7 *Formulae of multialgebraic specifications are of the following forms:*

1. *Atomic formulae, for terms $t, t' \in T(\Sigma, X)$:*
   - *$t \doteq t'$ (equality), t and t' denote the same one-element set.*
   - *$t \prec t'$ (inclusion), the set interpreting t is included in the set interpreting t'.*
2. *$a_1, \ldots, a_n \Rightarrow b_1, \ldots, b_m$, where either $n > 0$ or $m > 0$ and each a_i and b_j is atomic.*

Given a set of variables X, an assignment is a function $\alpha : X \to |A|$ assigning *individual* elements (not sets!) from the carrier of A to variables. In the standard way, it induces a unique interpretation $\overline{\alpha} : T(\Sigma, X) \to A$ of every term t (with variables from X) in A.

Satisfaction of formulae in a multialgebra is defined as follows:

Definition 2.8 *Given an assignment $\alpha : X \to |A|$:*

1. *$A \models_\alpha t \doteq t'$* iff $\overline{\alpha}(t) = \overline{\alpha}(t') = \{e\}$, for some element $e \in |A|$
2. *$A \models_\alpha t \prec t'$* iff $\overline{\alpha}(t) \subseteq \overline{\alpha}(t')$
3. *$A \models_\alpha a_1, \ldots, a_n \Rightarrow b_1, \ldots, b_m$* iff $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ or $\exists j : 1 \leq j \leq m : A \models_\alpha b_j$
4. *$A \models \varphi$* iff $A \models_\alpha \varphi$ for all α.

Multialgebraic specifications contain, as a special case, all usual equational specifications. The following example should give the flavor of some additional aspects of multialgebraic specifications.

Example 2.9 *A (sketchy) specification of the integers.*

$$\begin{array}{lllrl}
\multicolumn{5}{l}{\textbf{spec Int} =} \\
\mathbf{S}: & \multicolumn{4}{l}{Int} \\
\Omega: & zero: & & \to Int & \\
 & succ: & Int & \to Int & \\
 & pred: & Int & \to Int & \\
 & pos: & & \to Int & \\
\Phi: & 1. & & & zero \doteq zero \\
 & 2. & & & succ(x) \doteq succ(x) \\
 & 3. & & & pred(succ(x)) \doteq x \\
 & 4. & & & succ(zero) \prec pos \\
 & 5. & x \prec pos & \Rightarrow & succ(x) \prec pos
\end{array}$$

Axioms 1. and 2. state merely that zero and succ are deterministic operations. (Such axioms can be naturally abbreviated, e.g., by **det** : $zero, succ$*.) Axiom 3. states a property of pred applied to succ. Unlike the constant zero, the constant pos can be "nondeterministic", i.e., it can, in general, denote a set of elements of sort Int – its intention here is to represent the subsort of positive integers. The last two axioms specify the least contents of this subsort.*

Of course, a complete specification would include more axioms, but we merely wanted to indicate the special features of multialgebraic specifications.

Putting the definitions from this subsection together, we obtain the institution of multialgebras $\mathcal{MA}$. The property of $\mathcal{MA}$ which is of particular relevance for this paper follows from the following proposition by lemma 2.4.

Proposition 2.10 *The model functor* $\mathsf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *in the institution* $\mathcal{MA}$ *is finitely continuous, i.e.,* $\mathcal{MA}$ *is an exact institution.*

3 Specifications of parameterized data types

To specify parameterized data types we will use a restricted syntax for specifications. All specifications can be naturally viewed as standard multialgebraic specifications. Also, all semantic constructions take place in the category of standard multialgebras. Subsection 3.1 introduces merely convenient syntactic abbreviations.

3.1 Signatures with sort constants

We start by modifying the concept of signature and specification. The idea is that each signature may have, in addition to the standard set of sort and operation symbols, a (possibly empty) set of distinguished (sort and subsort) constant symbols $S^\star = \bigstar \cup C^\star$. The set $\bigstar$ contains constant symbols $\star_s$ for various sort symbols s – the intention of $\star_s$ is to denote all the elements of the respective sort s. The set $C^\star$ may contain additional constants which will represent various subsorts – the constants from this set are called "subsort constants". (Usually, the distinction between $\bigstar$ and $C^\star$ does not matter and then we will write "(sub)sort constants".)

Definition 3.1 *A signature with sort constants,* $\Sigma_\star$*, is a triple* $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$*, where* $\Sigma = (\mathbf{S}, \Omega)$ *is an ordinary signature and* $S^\star = \bigstar \cup C^\star$ *is a (possibly empty) set of additional constants,* $S^\star \cap \Omega = \emptyset$ *and* $\bigstar \cap C^\star = \emptyset$*.*

Signatures with sort constants will be used merely as a syntactic representation of ordinary signatures. This is possible in multialgebraic setting since constants may denote sets of elements. (In a more traditional setting, one would have to represent the sort constants, for instance, by predicates or (sub)sort symbols.)

We allow the set $S^\star$ to be empty. Also, we allow the set $\bigstar$ to contain several distinct constants of the same sort (although their intended meaning will be the same). The technical reasons for that come up in relating construction of co-limits (proposition 3.6, especially, lemma 3.9) and, in particular, pushouts (subsection 4.1.1). For the most, we think of the set $\bigstar$ as containing one constant for each sort, i.e., as $\bigstar = \{\star_s :\to s : s \in \mathbf{S}\}$. Most relevant constructions will involve and yield such signatures. (In general, we use the symbol $\star_s$ for an arbitrary constant from $\bigstar$ of sort s, e.g., for a signature morphism μ, $\mu(c) \neq \star_s$ means the same as $\mu(c) \notin \bigstar$.)

We will use the following operations relating the signatures with sort constants to ordinary signatures.

Definition 3.2 *Given a signature with sort constants $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$ we let:*

1. $\underline{\Sigma_\star} = (\mathbf{S}, \Omega \cup S^\star)$ *i.e.* $(\mathbf{S}, \Omega \cup \bigstar \cup C^\star)$ *– the underlying signature*
2. $\Sigma_- = (\mathbf{S}, \Omega, C^\star)$ *i.e.* $\Sigma_\star \setminus \bigstar$ *– the reduced signature*
3. $\Sigma = (\mathbf{S}, \Omega)$ *i.e.* $(\Sigma_\star \setminus S^\star)$ *– the standard (part of the) signature*

The other way around, given an ordinary signature $\Sigma = (\mathbf{S}, \Omega)$, we let

4. $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar)$ *, where $\bigstar = \{\star_s :\to s : s \in \mathbf{S}\}$ and $\bigstar \cap \Omega = \emptyset$ – the corresponding guarded signature with sort constants.*
5. $\Sigma^\uparrow = (\mathbf{S}, \Omega, \emptyset)$ *, – the included signature (with sort constants).*

Unless stated otherwise, the signatures considered will always be signatures with sort constants. Given an arbitrary specification **SP** we will sometimes write $\Sigma(\mathbf{SP})$ to denote its signature.

Definition 3.3 *A morphism between signatures with sort constants $\mu : \Sigma_\star \to \Sigma'_\star$ is a signature morphism between the underlying signatures $\mu : \underline{\Sigma_\star} \to \underline{\Sigma'_\star}$, sending $S^\star$ to $S^{\star\prime}$ and Σ to Σ'.*

In other words, the $\bigstar$-constants need not be sent to $\bigstar'$-constants but may be mapped to subsort constants $C^{\star\prime}$, as well.

Fact 3.4 *The signatures with sort constants form a category* $\mathbf{Sign}_\star$*, with the identity function as identity and function composition as composition.*

Since the signatures with sort constants essentially use underlying signature morphism, the transformation from the former to the latter can be extended to a functor $\_ : \mathbf{Sign}_\star \to \mathbf{Sign}$, which is the (pointwise) identity on the signature morphisms.

The other way around the inclusion of signatures (point 5 of def. 3.2) can also be extended to a functor $^\uparrow : \mathbf{Sign} \to \mathbf{Sign}_\star$, which sends Σ to $\Sigma^\uparrow$ and is the identity on morphisms. The following fact says that **Sign** can be treated as a full subcategory of $\mathbf{Sign}_\star$.

Fact 3.5 $^\uparrow : \mathbf{Sign} \to \mathbf{Sign}_\star$ *is full and faithfull.*

We have that for any $\Sigma \in \mathbf{Sign} : \Sigma = \underline{(\Sigma^\uparrow)}$ but, in general, for $\Sigma_\star \in \mathbf{Sign}_\star : \Sigma_\star \not\simeq (\underline{\Sigma_\star})^\uparrow$. This is because for an isomorphism in $\mathbf{Sign}_\star$ we must have isomorphism between the respective sets of (sub)sort constants, but while $S^\star$ in $\Sigma_\star$ may be non-empty, it is always empty in $(\underline{\Sigma_\star})^\uparrow$.

The following proposition shows that that (finite) co-limits in $\mathbf{Sign}_\star$ can be obtained from the respective co-limits in **Sign**.

Proposition 3.6 *The functor* $\_ : \mathbf{Sign}_\star \to \mathbf{Sign}$ *reflects (and preserves) finite co-limits.*

To prove the proposition we show that the functor reflects initial object, sums and co-equlizers.

Lemma 3.7 *The functor _ reflects (and preserves) initial objects.*

Proof. The empty signature $\Sigma^\emptyset = (\emptyset, \emptyset)$ is *the* initial object in **Sign**, and the empty signature $\Sigma^\emptyset_\star = (\emptyset, \emptyset, \emptyset)$ is *the* initial object in $\mathbf{Sign}_\star$. But $\Sigma^\emptyset = \underline{\Sigma^\emptyset_\star}$, so initial object is both reflected and preserved. □

Lemma 3.8 *The functor _ reflects (and preserves) sums (binary co-products).*

Proof. Let $\Sigma_{\star 1} = (\mathbf{S}_1, \Omega_1, S_1^\star)$ and $\Sigma_{\star 2} = (\mathbf{S}_2, \Omega_2, S_2^\star)$ be $\mathbf{Sign}_\star$ objects and T be a co-cone in $\mathbf{Sign}_\star$ as in the left diagram. Assume that its image $\underline{T}$ (as in the right diagram) is a co-limit (sum) in $\mathbf{Sign}$. We have to show that T is a sum in $\mathbf{Sign}_\star$.

$$\begin{tikzcd} T & \Sigma_\star & \\ \Sigma_{\star 1} \arrow[ur, "\iota_1"] & & \Sigma_{\star 2} \arrow[ul, "\iota_2"'] \end{tikzcd} \qquad \begin{tikzcd} \underline{T} & \underline{\Sigma_\star} \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}} & \\ \underline{\Sigma_{\star 1}} \arrow[ur, "\underline{\iota_1}"] & & \underline{\Sigma_{\star 2}} \arrow[ul, "\underline{\iota_2}"'] \end{tikzcd}$$

By the standard construction $\underline{\Sigma_\star} \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}}$, where the latter denotes disjoint union (of sort and operation symbols from both signatures). To simplify the notation, let us assume, without loss of generality, that we have equality here. Then both $\underline{\iota_i}$'s are injections,

Let C be any co-cone $\mu_1 : \Sigma_{\star 1} \to \Sigma'_\star$, $\mu_2 : \Sigma_{\star 2} \to \Sigma'_\star$ in $\mathbf{Sign}_\star$. Since $\underline{T}$ is a sum, we have a unique mediator to $\underline{C}$, $u_{\underline{\mu_1},\underline{\mu_2}} : \underline{\Sigma_\star} \to \underline{\Sigma'_\star}$, such that $\underline{\mu_i} = \underline{\iota_i}; u_{\underline{\mu_1},\underline{\mu_2}}$ for $i \in \{1, 2\}$. It is given by: for any symbol $x \in \underline{\Sigma_\star} : u_{\underline{\mu_1},\underline{\mu_2}}(x) = \begin{cases} \underline{\mu_1}(x) & if\ x \in \underline{\Sigma_{\star 1}} \\ \underline{\mu_2}(x) & if\ x \in \underline{\Sigma_{\star 2}} \end{cases}$.

$$\begin{tikzcd} C & \Sigma'_\star & \\ & \Sigma_\star \arrow[u, dotted, "u_{\mu_1,\mu_2}"] & \\ \Sigma_{\star 1} \arrow[uur, bend left, "\mu_1"] \arrow[ur, "\iota_1"] & & \Sigma_{\star 2} \arrow[ul, "\iota_2"'] \arrow[uul, bend right, "\mu_2"'] \end{tikzcd} \qquad \begin{tikzcd} \underline{C} & \underline{\Sigma'_\star} & \\ & \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}} \arrow[u, dotted, "u_{\underline{\mu_1},\underline{\mu_2}}"] & \\ \underline{\Sigma_{\star 1}} \arrow[uur, bend left, "\underline{\mu_1}"] \arrow[ur, "\underline{\iota_1}"] & & \underline{\Sigma_{\star 2}} \arrow[ul, "\underline{\iota_2}"'] \arrow[uul, bend right, "\underline{\mu_2}"'] \end{tikzcd}$$

The claim is that there is also a unique mediator $u_{\mu_1,\mu_2} : \Sigma_\star \to \Sigma'_\star$. Indeed, let it be given by $u_{\mu_1,\mu_2}(x) = u_{\underline{\mu_1},\underline{\mu_2}}(x)$ for all $x \in \Sigma_\star$ (then $\underline{u_{\mu_1,\mu_2}} = u_{\underline{\mu_1},\underline{\mu_2}}$). It obviously makes $\mu_i = \iota_i; u_{\mu_1,\mu_2}$, for $i \in \{1, 2\}$.

It is also a $\mathbf{Sign}_\star$ morphism because all ι_i's and μ_i's are: the (sub)sort constants $S_1^\star, S_2^\star$ from $\Sigma_{\star 1}, \Sigma_{\star 2}$, respectively, are mapped by ι_1, ι_2 to (sub)sort constants $S^\star$ in $\Sigma_\star$. Their ι_i images are also *all* the (sub)sort constants $S^\star$, since the other symbols in $\Sigma_\star$ are images of Σ_1, resp., Σ_2 symbols (i.e., of those symbols from $\Sigma_{\star 1}, \Sigma_{\star 2}$ which are *not* (sub)sort constants). Then, since also all the images under μ_i of (sub)sort constants from $\Sigma_{\star i}$ are (sub)sort constants in $\Sigma'_\star$, it follows from the definition of u_{μ_1,μ_2} that it, too, maps (sub)sort constants – of the form $c = \iota_i(c)$ – to (sub)sort constants, since $u_{\mu_1,\mu_2}(c) = \mu_i(c)$, for respective i's.

Finally, this u_{μ_1,μ_2} is unique making $\mu_i = \iota_i; u_{\mu_1,\mu_2}$. For if there is another $u \neq u_{\mu_1,\mu_2}$, such that $\mu_i = \iota_i; u$, then it would also be the case that $\underline{u} \neq u_{\underline{\mu_1},\underline{\mu_2}}$ and, furthermore, that $\underline{\mu_i} = \underline{\iota_i}; \underline{u}$, contradicting the uniqueness of $u_{\underline{\mu_1},\underline{\mu_2}}$.

Preservation of sums follows now easily. A sum $\Sigma_{\star 1} + \Sigma_{\star 2}$ in $\mathbf{Sign}_\star$ must be isomorphic to a sum as given above, i.e., a disjoint union of all the symbols from both signatures: $\Sigma_{\star 1} + \Sigma_{\star 2} \simeq (\mathbf{S}_1 \uplus \mathbf{S}_2, \Omega_1 \uplus \Omega_2, S_1^\star \uplus S_2^\star)$ where also $(\Omega_1 \uplus \Omega_2) \cap (S_1^\star \uplus S_2^\star) = \emptyset$. But then $\underline{\Sigma_{\star 1} + \Sigma_{\star 2}} \simeq (\mathbf{S}_1 \uplus \mathbf{S}_2, (\Omega_1 \cup S_1^\star) \uplus (\Omega_2 \cup S_2^\star)) \simeq \underline{\Sigma_{\star 1}} + \underline{\Sigma_{\star 2}}$. □

Lemma 3.9 *The functor _ reflects (and preserves) co-equalizers.*

Proof. Let $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$, $\Sigma'_\star = (\mathbf{S}', \Omega', S^{\star\prime})$, $\Sigma''_\star = (\mathbf{S}'', \Omega'', S^{\star\prime\prime})$ be objects in $\mathbf{Sign}_\star$ and suppose that we have a co-cone in $\mathbf{Sign}_\star$ as on the left diagram (with $\mu_1; \sigma = \mu_2; \sigma$), and that its image (on the right diagram) is a co-limit (co-equalizer) in $\mathbf{Sign}$. We have to show that the original co-cone (on the left) is a co-limit in $\mathbf{Sign}_\star$.

$$\begin{tikzcd} \Sigma_\star \arrow[r, shift left, "\mu_1"] \arrow[r, shift right, "\mu_1"'] & \Sigma'_\star \arrow[r, "\sigma"] & \Sigma''_\star \end{tikzcd} \qquad \begin{tikzcd} \underline{\Sigma_\star} \arrow[r, shift left, "\underline{\mu_1}"] \arrow[r, shift right, "\underline{\mu_1}"'] & \underline{\Sigma'_\star} \arrow[r, "\underline{\sigma}"] & \underline{\Sigma''_\star} \simeq \underline{\Sigma'_\star}/\!\approx \end{tikzcd}$$

By the standard construction in **Sign**, we have an isomorphism $\underline{\Sigma''_\star} \simeq \underline{\Sigma'_\star}/_\approx$, where $\underline{\Sigma'_\star}/_\approx = (\mathbf{S}'/_\approx, \Omega'/_\approx)$, is the standard choice of co-equalizer i.e. the quotient by the least equivalence $\approx$ on $\underline{\Sigma'_\star}$ induced by the relation with the following components:

1. Sorts: $\approx_{\mathbf{S}'} = \{\langle \underline{\mu_1}(s), \underline{\mu_2}(s)\rangle : s \in \mathbf{S}\}$,
2. Operations: $\approx_{\Omega' \cup S^{\star\prime}} = \{\langle \underline{\mu_1}(\omega), \underline{\mu_2}(\omega)\rangle : \omega \in \Omega \cup S^\star\}$

To simplify the notation we will assume, without loss of generality, that, in fact, $\underline{\Sigma''_\star} = \underline{\Sigma'_\star}/_\approx$.

Since μ_1, μ_2 are $\mathbf{Sign}_\star$-morphisms, the equivalence $\approx$ above can be viewed (is the same) as the least equivalence $\sim$ on $\Sigma'_\star$ induced by the following components:

3. Sorts: $\sim_{\mathbf{S}'} = \{\langle \mu_1(s), \mu_2(s)\rangle : s \in \mathbf{S}\}$,
4. Operations: $\sim_{\Omega'} = \{\langle \mu_1(\omega), \mu_2(\omega)\rangle : \omega \in \Omega\}$
5. (sub)sort constants: $\sim_{S^{\star\prime}} = \{\langle \mu_1(c), \mu_2(c)\rangle : c \in S^\star\}$.

We then have $\underline{\Sigma'_\star/_\sim} = \underline{\Sigma'_\star}/_\approx$, so we let $\Sigma''_\star = \Sigma'_\star/_\sim$.

Let μ_1, μ_2, γ be an arbitrary co-cone as shown on the left diagram.

$$\begin{tikzcd} \Sigma_\star \arrow[r, shift left, "\mu_1"] \arrow[r, shift right, "\mu_1"'] & \Sigma'_\star \arrow[r, "\sigma"] \arrow[dr, "\gamma"'] & \Sigma''_\star = \Sigma'_\star/_\sim \arrow[d, dotted, "u_\gamma"] \\ & & \Sigma'''_\star \end{tikzcd} \qquad \begin{tikzcd} \underline{\Sigma_\star} \arrow[r, shift left, "\underline{\mu_1}"] \arrow[r, shift right, "\underline{\mu_1}"'] & \underline{\Sigma'_\star} \arrow[r, "\underline{\sigma}"] \arrow[dr, "\underline{\gamma}"'] & \underline{\Sigma'_\star}/_\approx \arrow[d, dotted, "u_{\underline{\gamma}}"] \\ & & \underline{\Sigma'''_\star} \end{tikzcd}$$

Since the image $\underline{\sigma}$, $\underline{\Sigma''_\star} = \underline{\Sigma'_\star/_\sim} = \underline{\Sigma'_\star}/_\approx$ is co-equalizer, we have a unique mediator $u_{\underline{\gamma}}$ making $\underline{\gamma} = \underline{\sigma}; u_{\underline{\gamma}}$. We show that $u_\gamma : \Sigma''_\star \to \Sigma'''_\star$, given by $u_\gamma(x) = u_{\underline{\gamma}}(x)$ for all symbols $x \in \Sigma''_\star$ (in particular, $\underline{u_\gamma} = u_{\underline{\gamma}}$) is a unique mediator in $\mathbf{Sign}_\star$. It obviously makes $\gamma = \sigma; u_\gamma$.

It is also a morphism in $\mathbf{Sign}_\star$. Since both σ, γ are morphisms in $\mathbf{Sign}_\star$ they send (sub)sort constants $S^{\star\prime}$ to the (sub)sort constants $S^{\star\prime\prime}$, respectively, $S^{\star\prime\prime\prime}$. Since $\underline{\sigma}$ is surjective, then so is σ, and thus the (sub)sort constants in $\Sigma''_\star$ are exactly the σ-images of (sub)sort constants $S^{\star\prime}$ from $\Sigma'_\star$. By definition of u_γ and the fact that $\gamma = \sigma; u_\gamma$, this means that for every (sub)sort constant $\sigma(c) = [c] \in S^{\star\prime\prime}$, $u_\gamma([c]) = \gamma(c) \in S^{\star\prime\prime\prime}$.

Finally, u_γ is a unique mediator. For if there was another $u \neq u_\gamma$ making $\gamma = \sigma; u$, then we would also have $\underline{u} \neq u_{\underline{\gamma}}$ and $\underline{\gamma} = \underline{\sigma}; \underline{u}$, contradicting the uniqueness of $u_{\underline{\gamma}}$.

The fact that co-equalizers are also preserved by the functor _ follows now easily. A co-equalizer of μ_1, μ_2 must be isomorphic to the quotient $\Sigma'_\star/_\sim$, with $\sim$ defined by the points 3.-5. above. But then its image $\underline{\Sigma'_\star/_\sim}$ is trivially isomorphic to the co-equalizer $\underline{\Sigma'_\star}/_\approx$ in **Sign**. $\square$

Since we can create all finite co-limits by initial objects, sums and co-equlizers the proposition 3.6 follows from the above lemmata. The concrete way of doing this is to construct a co-limit in **Sign** and then make an appropriate choice of the (sub)sort constants, according to the prescriptions given in the proofs above.

3.2 Guarded specifications

To write specifications of parameterized data types we will use guarded axioms. In general, one only requires that the axioms from the parameter specification hold only for elements in the parameter algebras, and guards are needed to mark these elements.

Definition 3.10 *Given a signature $\Sigma_\star$ (with sort constants $S^\star$):*

1. *a* guard *γ is an atom of the form $x \prec c$, where x is a variable and $c \in S^\star$;*
2. *a* guarded formula *is of the form $\phi_\star = \gamma^*, \overline{a} \Rightarrow \overline{b}$ where $\overline{a}, \overline{b}$ are (sequences of) Σ_- atoms and γ^* is a (non-empty) sequence of guards $\gamma_i = x_i \prec c_i$ for some variables x_i occuring in the atoms $\overline{a}, \overline{b}$ – γ_i's in such conditional formulae are called* local guards*;*
3. *a* fully guarded *formula is a guarded formula where there is a guard γ_i for each variable x_i occurring in the formula;*

4. an unguarded formula *is any formula over* $\Sigma_\star$ *with no guards.*

The only places where $\bigstar$ may occur are in guards – for conditional axioms in the premises. (We will also allow unconditional axioms of form 1 and 4.). Note that guards $x \prec \star$ are only special (strongest) cases – in general, $x \prec c$, where $c \in S^\star$, is a guard.

A formula containing only ground $\Sigma_\star$ terms (a ground $\Sigma_\star$ formula) can be seen either as a fully guarded formula (according to point 3.), or else as an unguarded formula (according to point 4.). Saying "a fully guarded formula" we will typically mean also such formulae.

We will use only a restricted form of the specifications, namely, guarded specifications.

Definition 3.11 *A* guarded specification *is a triple* $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$ *where:*

- $\Sigma_\star$ *is a signature with sort constants*
- *Each* $\phi_\star \in \Phi_\star$ *is a guarded or unguarded formula*
- $\Gamma_\Sigma \subseteq \{x \prec \star_s : \star_s \in \bigstar\}$ *is the set of axioms called* global guards.

If $\bigstar$ *contains at least one constant of each sort and there is an equality in the last point, we call a specification* globally guarded*. If, in addition, all axioms in* $\Phi_\star$ *are fully guarded (possibly, ground unguarded* $\Sigma_\star$ *formulae), the specification is* fully guarded.

Note that all the local guards of the form $x \prec \star \Rightarrow ...$ in a globally guarded specification are trivially satisfied due to the presence of the global guards Γ.

Example 3.12 *Fully guarded specification of the natural numbers.*

$$
\begin{array}{lllll}
\multicolumn{5}{l}{\textbf{spec Nat}_\star =} \\
\mathbf{S}: & Nat & & & \\
\Omega: & zero: & \to Nat & & \\
 & succ: & Nat \to Nat & & \\
 & pred: & Nat \to Nat & & \\
S^\star: & \star_{Nat}: & \to Nat & & \\
\Phi: & 1. & & & zero \doteq zero \\
 & 2. & x \prec \star_{Nat} & \Rightarrow & succ(x) \doteq succ(x) \\
 & 3. & x \prec \star_{Nat} & \Rightarrow & pred(succ(x)) \doteq x \\
\Gamma: & 4. & & & x \prec \star_{Nat}
\end{array}
$$

Obviously, there is no real difference between this and the usual specification of **Nat** (except for the constant $\star_{Nat}$ which comprises all the elements of sort Nat.)

Definition 3.13 *Given a guarded specification* $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$*,*

1. *its* weakening *is a specification* $\mathbf{SP}_- = (\Sigma_\star, \Phi_\star)$.
2. *its* underlying *specification is* $\underline{\mathbf{SP}_\star} = (\underline{\Sigma_\star}, \Phi_\star \cup \Gamma_\Sigma)$.

Conversely, for an ordinary specification $\mathbf{SP} = ((\mathbf{S}, \Omega), \Phi)$*,* $\mathbf{SP}_\star$ *denotes the fully guarded specification* $(\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$*, where* $\bigstar$ *is as in def. 3.2,* $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar)$ *and* $\Gamma_\Sigma = \{x \prec \star : \star \in \bigstar\}$.

As for formulae, $\mathbf{SP}_\star$ denotes, in general, a guarded specification with all, some or none axioms guarded. However, when $\mathbf{SP}$ is an ordinary specification, then $\mathbf{SP}_\star$ denotes the fully guarded version. This latter case will be used less frequently than the former one, and we will indicate it explicitly.

Keep also in mind that, given a specification $\mathbf{SP}_\star$ with signature $\Sigma_\star$, the signature of its weakened version $\mathbf{SP}_-$ is still $\Sigma_\star$ and not Σ_-.

As models for guarded specifications we use ordinary multialgebras.

Definition 3.14 *The model class of a guarded specification* $\mathbf{SP}_\star$ *is the model class of its underlying specification:* $\mathsf{Mod}(\mathbf{SP}_\star) = \mathsf{Mod}(\underline{\mathbf{SP}_\star})$.

In particular, for a given ordinary specification $\mathbf{SP}$, there is obvious equivalence of model categories between the unguarded $\mathsf{Mod}(\mathbf{SP})$ and fully guarded $\mathsf{Mod}(\mathbf{SP}_\star)$. Also, for a given guarded specification $\mathbf{SP}_\star$ there is the obvious inclusion functor

$$\mathsf{id}_- : \mathsf{Mod}(\mathbf{SP}_\star) \to \mathsf{Mod}(\mathbf{SP}_-) \tag{2}$$

which sends each algebra in the first class to itself.

3.3 Syntax of parameterized data type specification

Definition 3.15 *A parameterized data type specification (a PDT) is a quadruple* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, *where*

1. $\mathbf{X}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$ *and* $\mathbf{P}[\mathbf{X}]_\star = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'})$ *are fully guarded specifications,*
2. $\Sigma_\star = (\mathbf{S}, \Omega, S^\star) \subseteq (\mathbf{S}', \Omega', S^{\star\prime}) = \Sigma'_\star$ *are signatures from* $\mathbf{Sign}_\star$,
3. $\mu : \Sigma_\star \to \Sigma'_\star$ *is a signature morphism, called the* "parameterization morphism", *which is identity on all symbols with the possible exception of* $\bigstar$, *and such that if* $\mu(\star_s) \neq \star_s$ *then* $\mu(\star_s) \notin \Sigma_\star$ *(i.e.,* $\mu(\star_s)$ *is then a fresh constant from* $S^{\star\prime}$*),*
4. $\delta : \Sigma_\star \to \Sigma'_\star$, *called the* "local guard mapping", *is a signature morphism, which is identity on all symbols with the possible exception of* $\bigstar$,
5. *the two specifications,* μ *and* δ *are such that:*
   (a) *if* $\mu(\star_s) \neq \delta(\star_s)$ *then the axiom* $\mu(\star_s) \prec \delta(\star_s)$ *is among* $\Phi'_\star$
   (b) *for every unguarded axiom* $\phi \in \Phi_\star : \mu(\phi) \in \Phi'_\star$
   (c) *for every guarded axiom* $\phi_\star \in \Phi_\star : \delta(\phi_\star) \in \Phi'_\star$.

We write δ at the end of the tuple because in many constructions it plays no role. In such cases, we will often drop it from the notation and write a PDT simply as $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$. Although it is defined as a signature morphism, its only role is to map the local guards. Being the identity on most symbols, it is essentially a mapping $\delta : \bigstar \to S^{\star\prime}$. The point 5c can be stated in more detail as follows: for every $\phi_\star \in \Phi_\star$:
$\phi_\star = x_1 \prec \star_1, \ldots, x_m \prec \star_m, \overline{a} \Rightarrow \overline{b}$,
where $x_1 \prec \star_1, \ldots, x_m \prec \star_m$ are all the local guards in $\phi_\star$ with $\star_i \in \bigstar$ (and $\overline{a}, \overline{b}$ sequences of Σ_- atoms), $\delta(\phi_\star) \in \Phi'_\star$, where $\delta(\phi_\star)$ is the corresponding axiom:
$\delta(\phi_\star) = x_1 \prec \delta(\star_1), \ldots, x_m \prec \delta(\star_m), \overline{a} \Rightarrow \overline{b}$.
Similarly, μ is essentially a mapping $\mu : \bigstar \to S^{\star\prime}$. Although the definition is rather liberal, for all practical purposes we can think of the syntax as given by a μ which maps $\bigstar$ to $S^{\star\prime}$, and by a δ with $\delta(\star_s) = \mu(\star_s)$ or else $\delta(\star_s) = \star_s$ (see the remarks below). This, in fact, covers most natural situations and will be the case in all our examples.

Axioms of the form 5a are needed to ensure that the guarded axioms from the parameter specification will still apply, at least, to the elements originating from the parameter algebra. Since $\mathbf{X}_\star$ is fully guarded, axioms mentioned in 5b are actually ground formulae, typically, involving only (sub)sort constants from $S^\star$. The restriction 5b has significant effect if, e.g., ϕ is $c \prec \star$ and $\mu(\star) = c' \neq \star$, in which case, we require $\mu(\phi) = c \prec c'$ to be included in $\Phi'_\star$. This requirement ensures that elements contained in c (in a parameter algebra) will remain within $\mu(\star)$ (we will see it shortly when discussing the semantics).

The definition discriminates against some isomorphic (wrt. semantics) specifications, since μ has to be identity on Σ_-. It will be the main element in defining the semantics of PDTs. The image under μ of the constants $\bigstar$ can be twofold – $\mu(\star_s)$ has to be $\star_s$ itself or a fresh constant from $C^{\star\prime}$.

1) The former situation, $\mu(\star_s) = \star_s$, corresponds to the classical case of persistency, i.e., to non-extending the carrier.

2) Otherwise, $\mu(\star_s) \neq \star_s$, we will obtain a distinction between the elements of sort s originating from the parameter, $\mu(\star_s)$, and the ones originating from the parameterized specification itself, the "new" $\star_s$.

These two cases can be marked by the respective keywords: 1) **non-extending carrier** s, or 2) (possibly) **extending carrier** s.

The mapping δ is introduced to allow more flexibility in parameterized data type specifications. If the carrier of sort s is not extended, case 1) above, δ has no effect – according to 5a, it has to be $\delta(\star_s) = \mu(\star_s) = \star_s$ (or else another subsort constant c but such that $\mathbf{P}[\mathbf{X}]_\star \models \star_s \prec c$ which, together with the global guard $x \prec \star_s$ says that the two denote the same set, i.e., the whole carrier of sort s). But if the carrier of s is (possibly) extended, case 2) above, δ allows to either

2a) restrict the local guards from the formal parameter, when $\delta(\star_s) = \mu(\star_s)$ – in this case the axioms from the parameter specification are required to hold only for the elements from the parameter algebra,

2b) or else extend the local guards – in which case the axioms from the parameter specification have to hold also for (possibly new) elements from $\delta(\star_s)$; the presence of axioms 5a, $\mu(\star_s) \prec \delta(\star_s)$, ensures that the old axioms still hold at least for the old elements.

Thus, in the case 2) of **extending carrier** s, δ allows us either 2a) **restricting (local) guards** s, or 2b) **extending (local) guards** s.

Example 3.16 *Specification of groups parameterized by monoids.*

First, we take a standard (deterministic multialgebraic) specification of groups, with multiplication $\cdot$, unit e and inverse $(\_)^-$:

$$
\begin{array}{lrrcl}
\textbf{spec Group} = & & & & \\
\mathbf{S}': & S & & & \\
\Omega': & \cdot: & S \times S & \to & S \\
 & e: & & \to & S \\
 & (\_)^-: & S & \to & S \\
\Phi': & 1. & e \cdot x & \doteq & x \cdot e \\
 & 2. & e \cdot x & \doteq & x \\
 & 3. & x \cdot (y \cdot z) & \doteq & (x \cdot y) \cdot z \\
 & 4. & e & \doteq & e \\
 & 5. & x \cdot y & \doteq & x \cdot y \\
 & 6. & x \cdot x^- & \doteq & e \\
 & 7. & x^- & \doteq & x^-
\end{array}
$$

This does not conform to the required format – we add the sort constant $\star_S$, a global guard $x \prec \star_S$, and then a new constant mono to mark the elements originating from the monoid specification.

The result is the PDT on the right, with the formal parameter on the left:

$$
\boxed{\begin{array}{lllcl}
\multicolumn{5}{l}{\textbf{spec Monoid}_\star =} \\
\mathbf{S}: & S & & & \\
\Omega: & \cdot: & S\times S \to S & & \\
 & e: & \to S & & \\
S^\star: & \star_S: & \to S & & \\
\Phi: & 1. & e\cdot x & \doteq & x\cdot e \\
 & 2. & e\cdot x & \doteq & x \\
 & 3. & x\cdot(y\cdot z) & \doteq & (x\cdot y)\cdot z \\
 & 4. & e & \doteq & e \\
 & 5. & x\cdot y & \doteq & x\cdot y \\
\Gamma: & 6. & x & \prec & \star_S
\end{array}}
\quad \xrightarrow[\delta(\star_S)=\star_S]{\mu(\star_S)=mono} \quad
\boxed{\begin{array}{lllcl}
\multicolumn{5}{l}{\textbf{spec Group[Monoid]}_\star =} \\
\mathbf{S}': & S & & & \\
\Omega': & \cdot: & S\times S \to S & & \\
 & e: & \to S & & \\
 & (\_)^-: & S \to S & & \\
S^{\star\prime}: & \star_S: & \to S & & \\
 & mono: & \to S & & \\
\Phi': & 1. & e\cdot x & \doteq & x\cdot e \\
 & 2. & e\cdot x & \doteq & x \\
 & 3. & x\cdot(y\cdot z) & \doteq & (x\cdot y)\cdot z \\
 & 4. & e & \doteq & e \\
 & 5. & x\cdot y & \doteq & x\cdot y \\
 & 6. & x\cdot x^- & \doteq & e \\
 & 7. & x^- & \doteq & x^- \\
\Gamma': & 8. & x & \prec & \star_S
\end{array}}
$$

The parameterization morphism sends $\mu(\star_S) = mono$. This is so because constructing a **Group**-*algebra out of a given* **Monoid**-*algebra, one may need to add new elements,* **extending carrier**, *of sort S. The (dummy) constant mono keeps the track of the old elements.*

On the other hand, even if new elements appear in the resulting algebra, we certainly want the **Monoid**-*axioms to hold not only for the old but also for these new elements. Therefore, we let $\delta(\star_S) = \star_S$.*

In the presentation above, we have dropped all the local guards $x \prec \star_S \Rightarrow ...$ in both specifications, since they will be trivially satisfied due to the presence of the global guards Γ, resp. Γ'. We have also dropped the axiom $mono \prec \star_S$ (required by definition 3.15, point 5a), since it follows from the global guard Γ'. The respective specifications are equivalent (i.e., isomorphic in $\mathbf{Th}_{\mathcal{MA}}$). We will often use such abbreviations in order to simplify the examples. (The syntactic presence of all guards will be of significance first in considering composition in section 5.)

It should be observed that the flexibility offered by δ is highly desirable – in many other situations, one would like to restrict the local guards from the formal parameter to hold only for the elements originating from it but not for the new ones added by the parameterized specification. The classical example of such a situation is a specification of stacks parameterized by elements.

Example 3.17 *As in the previous example, we drop the local guards $s \prec \star_{Stack}$ and the axiom $ok \prec \star_{El}$ from* $\mathbf{Stack[El]}_\star$.

$$
\boxed{\begin{array}{ll}
\multicolumn{2}{l}{\textbf{spec El}_\star =} \\
\mathbf{S}: & El \\
S^\star: & \star_{El} :\to El \\
\Gamma: & x \prec \star_{EL}
\end{array}}
\quad \xrightarrow[\delta(\star_{El})=ok]{\mu(\star_{El})=ok} \quad
\boxed{\begin{array}{lllll}
\multicolumn{5}{l}{\textbf{spec Stack[El]}_\star =} \\
\mathbf{S}': & Stack, El & & & \\
\Omega': & empty: & \to Stack & & \\
 & top: & Stack \to El & & \\
 & pop: & Stack \to Stack & & \\
 & push: & El\times Stack \to Stack & & \\
S^{\star\prime}: & \star_{El}: & \to El & & \\
 & \star_{Stack}: & \to Stack & & \\
 & ok: & \to El & & \\
\Phi': & 1. & & & empty \doteq empty \\
 & 2. & x \prec ok & \Rightarrow & top(push(x,s)) \doteq x \\
 & 3. & x \prec ok & \Rightarrow & pop(push(x,s)) \doteq s \\
\Gamma': & 4. & & & x \prec \star_{EL} \\
 & 5. & & & s \prec \star_{Stack}
\end{array}}
$$

The parameterization morphism with $\mu(\star_{El}) = ok$, allows **extending carrier** *of sort El. The local guard mapping δ coincides here with μ, i.e., $\delta(\star_{El}) = ok$, thus* **restricting (local) guards** *of sort El – the potentially new elements of sort El arising in* **Stack**, *like top(empty), are not intended to behave as the "ordinary" elements from the actual parameter sort but to function merely as, say, "error" elements.*

The effect of δ may be little visible in the above example since the parameter $\mathbf{El}_\star$ does not contain any proper axioms. However, it should be clear that a PDT where the local guards in $\mathbf{Stack}[\mathbf{El}]_\star$ axioms 2. and 3. were replaced by $x \prec \star_{El}$ would be very different. Also, it should be easy to imagine replacing the formal parameter $\mathbf{El}_\star$ with, say $\mathbf{Nat}_\star$ from example 3.12, in which axioms should be guarded. For instance, the axiom $x \prec \star_{Nat} \Rightarrow pred(succ(x)) \doteq x$ from $\mathbf{Nat}_\star$ would correspond to $x \prec ok \Rightarrow pred(succ(x)) \doteq x$ in $\mathbf{Stack}[\mathbf{Nat}]_\star$, since we do not necessarily want this axiom to apply to the result of $top(empty)$.

We register a simple fact about the parameterization morphisms:

Fact 3.18 *If $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ is a parameterized data type specification then:*

1. *$\mu : \mathbf{X}_\star \to \mathbf{P}[\mathbf{X}]_\star$ may not be a specification morphism, but*
2. *$\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_-$ is a specification morphism, and hence also*
3. *$\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_\star$ is a specification morphism.*

Proof. 1. A trivial counter-example is given by $\mathbf{X}_\star$ with the only global guard axiom $x \prec \star$, $\mathbf{P}[\mathbf{X}]_\star$ with a subsort constant c and the only global guard axiom $x \prec \star$, and $\mu(\star) = \delta(\star) = c$. Obviously $\mathbf{P}[\mathbf{X}]_\star \not\models \mu(x \prec \star) = x \prec c$.

2. follows from definition 3.15. μ-translations of all unguarded axioms from $\mathbf{X}_-$ are included in $\mathbf{P}[\mathbf{X}]_\star$ by point 5b. For any guarded axiom $x_i \prec \star_i, \overline{a} \Rightarrow \overline{b}$ from $\mathbf{X}_-$, the corresponding axiom $x_i \prec \delta(\star_i), \overline{a} \Rightarrow \overline{b}$ is in $\mathbf{P}[\mathbf{X}]_\star$. But then we also have the axioms $\mu(\star_i) \prec \delta(\star_i)$ in $\mathbf{P}[\mathbf{X}]_\star$, and these together imply that $\mathbf{P}[\mathbf{X}]_\star \models x_i \prec \mu(\star_i), \overline{a} \Rightarrow \overline{b}$, i.e., $\mathbf{P}[\mathbf{X}]_\star \models \mu(x_i \prec \star_i, \overline{a} \Rightarrow \overline{b})$.

3. follows from 2., since $\mathbf{P}[\mathbf{X}]_\star \models \mathbf{P}[\mathbf{X}]_-$. □

3.4 Semantics of parameterized data type specification

It is, of course, possible to use ordinary (loose) semantics for our PDTs, i.e., to treat them as simple parameterized specifications. But the trouble we have taken with the syntactic operations and restrictions on the specifications was meant to provide the possibility to define the semantics as parameterized data types, i.e., data types consisting of algebras parameterized by algebras. It seems to us satisfying that PDTs can be seen as more specific, special cases of parameterized specifications.

To define the semantics for PDTs we will use a special case of the general (weak) homomorphisms of multialgebras.

Definition 3.19 *A* tight *homomorphism $h : A \to B$ is a homomorphism satisfying:*

$$h(\omega^A(x_1...x_n)) = \omega^B(h(x_1)...h(x_n))$$

The tight homomorphisms and, in particular, tight monomorphisms have a logical counterpart, which will be useful in some proofs and which we now establish in proposition 3.22.

Lemma 3.20 *Let $A, B \in \mathsf{Mod}(\Sigma_\star)$, $\iota : A \to B$ be a tight $\Sigma_\star$-homomorphism, $t(x_1...x_n)$ be a $\Sigma_\star$ term, and $\alpha : \{x_1...x_n\} \to |A|$ an assignment. Then $\iota(t^A(\alpha(x_1)...\alpha(x_n))) = t^B(\iota(\alpha(x_1))...\iota(\alpha(x_n)))$.*

Proof. For any constant c we have $\iota(c^A) = c^B$ by definition. We drop multiple arguments to simplify notation. For any oparation ω, we have $\iota(\omega^A(\alpha(x))) = \omega^B(\iota(\alpha(x)))$. The result for any term t follows trivially by induction. □

Corollary 3.21 *With the notation from the previous lemma, let ι be a tight monomorphism.*

- *For any assignment $\alpha : X \to |A|$, let $\beta : X \to |B|$ be given by $\alpha;\iota$.*
- *Conversely, for any assignment $\beta : X \to |B|$, such that $\forall x \in X : \beta(x) \in \iota[A]$, let $\alpha : X \to |A|$ be given by $\beta;\iota^-$.*

Then for any atomic formula $a : A \models_\alpha a \iff B \models_\beta a$.

Proof. To simplify the notation, we write only a single argument/variable $X = \{x\}$.

1. If $A \models_\alpha s(x) \preceq t(x)$ (where $\preceq$ stands for $\prec$ or $\doteq$), then by 3.20, $s^B(\iota(\alpha(x))) = \iota(s^A(\alpha(x))) \preceq \iota(t^A(\alpha(x))) = t^B(\iota(\alpha(x)))$.

Conversely, if $B \models_\beta s(x) \doteq t(x)$, i.e., if $s^B(\iota(\alpha(x))) \doteq t^B(\iota(\alpha(x)))$, then by 3.20, $\iota(s^A(\alpha(x))) = \iota(t^A(\alpha(x)))$. But since ι is mono (i.e., injective, [15]), this means that $\iota(s^A(\alpha(x))) \doteq \iota(t^A(\alpha(x)))$, and so $s^A(\alpha(x)) \doteq t^A(\alpha(x))$, i.e., $A \models_\alpha s(x) \doteq t(x)$.

If $B \models_\beta s(x) \prec t(x)$, i.e., if $s^B(\iota(\alpha(x))) \subseteq t^B(\iota(\alpha(x)))$ then, by the same argument (in particular, injectivity of ι), we get $s^A(\alpha(x))) \subseteq t^A(\alpha(x)))$, i.e., $A \models_\alpha s(x) \prec t(x)$.

2. Notice, that since ι is a monomorphism, $\alpha = \beta;\iota^-$ in point 2. is well defined. But then, $\alpha;\iota = \beta$ and the result follows by point 1. $\square$

Proposition 3.22 *With the notation from the above above corollary, let ι be a tight monomorphism and let ϕ be an arbitrary, fully guarded formula. Then $A \models \phi \iff B \models \phi$.*

Proof. Again, to simplify the notation, we write only a single argument/variable $X = \{x\}$. Let ϕ be $x \prec c, a_1, ..., a_m \Rightarrow a_{m+1}, ..., a_n$.

Assume that $A \models \phi$ and let $\beta : X \to |B|$ be arbitrary assignment. If $\beta(x) \notin c^B$, then $B \models \phi$. If, on the other hand, $\beta(x) \in c^B$, then we can define $\alpha : X \to |A|$, as in point 2. of corollary 3.21. But then we get that for all $a_i : A \models_\alpha a_i \iff B \models_\beta a_i$, i.e., $B \models_\beta \phi$. Since β was arbitrary, it follows that $B \models \phi$.

So assume that $A \not\models \phi$, and let $\alpha : X \to |A|$ be an assignment falsifying ϕ. Defining β as in point 1. of corollary 3.21, we get $B \not\models_\beta \phi$, i.e., $B \not\models \phi$. $\square$

The above result will only appear in some proofs later on.

Now, given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ and a functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, we obtain two functors:

- $\mathsf{id}_\_ : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_\_)$ defined in (2) at the end of section 3.1, and
- the composition $\mathsf{F};\_|_\mu : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_\_)$.

The latter has the target $\mathsf{Mod}(\mathbf{X}_\_)$ and not $\mathsf{Mod}(\mathbf{X}_\star)$ because, in the case when $\mu(\star_s) = c \neq \star_s$ for some s, the reduct $A|_\mu$ of an algebra $A \in \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ may contain more elements in the sort $s^{A|_\mu}$ than those in $\star_s^{A|_\mu}$, i.e., it may fail to satisfy the global guard $x \prec \star_s$. This captures the intention that the parameterized algebra may actually add new elements to the sorts of the parameter algebra. We define (loose) semantics of parameterized data type specifications by putting some restrictions on the functors $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$. Notice that the effect of (or requirements put by) δ are present in the actual axioms of both specifications, so that we do not have to consider δ explicitly here.

Definition 3.23 *The semantics of the parameterized data type specification $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ is the class of all functors $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, such that there exists a natural transformation $\iota : \mathsf{id}_\_ \rightrightarrows \mathsf{F};\_|_\mu$, where for each $A \in \mathsf{Mod}(\mathbf{X}_\star)$ the component ι_A is a tight $\Sigma(\mathbf{X}_\star)$-monomorphism.*

A functor with this property is called a *semantic functor* for the PDT $\mathbf{P}$, and $\mathsf{PMod}(\mathbf{P})$ will denote the class of all semantic functors for a parameterized data type.

First, let us observe that although this semantics $\mathsf{PMod}(\mathbf{P})$ is rather liberal, it does capture the structuring aspect at least in the sense that the algebras which may result from it are not all possible models of the flat specification. More precisely, a PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ can be seen as defining two classes of algebras:

1. the one is simply the model class $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ with $\mathbf{P}[\mathbf{X}]_\star$ viewed as a flat specification, and
2. the other is obtained from the semantic functors of the PDT, namely, the class $\mathsf{FMod}(\mathbf{P}) = \{\mathsf{F}(X) : X \in \mathsf{Mod}(\mathbf{X}_\star), \mathsf{F} \in \mathsf{PMod}(\mathbf{P})\}$ of all $\mathbf{P}[\mathbf{X}]_\star$ algebras which can be obtained as an image of some $\mathbf{X}_\star$ algebra under some semantic functor F.

Obviously, the latter class is contained in the former.

Fact 3.24 *In general* $\mathsf{FMod}(\mathbf{P}) \neq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$.

Proof. Consider following specification $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$:

$$
\begin{array}{lllll}
\textbf{spec } \mathbf{X}_\star = & & & & \textbf{spec } \mathbf{P}[\mathbf{X}]_\star = \\
\mathbf{S}: & s & & & \mathbf{S}': \quad s \\
\Omega: & c: & \to s & \xrightarrow{\mu(\star_s)=d} & \Omega': \quad c: \quad \to s \\
S^\star: & \star_s: & \to s & & S^{\star\prime}: \quad d, \star_s: \quad \to s \\
\Gamma: & 1. & x \prec \star_s & & \Gamma': \quad 1. \qquad x \prec \star_s
\end{array}
$$

The algebra A given by: carrier $|A| = \{c\}$ and $c^A = \star_s^A = c, d^A = \emptyset$, is in $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$. However, the reduct $A|_\mu$ has no tight $\Sigma_\star$-subalgebra in $\mathsf{Mod}(\mathbf{X}_\star)$, because $\star_s^{A|_\mu} = \emptyset$ while $c^{A|_\mu} = c$. Hence $\mathsf{FMod}(\mathbf{X}) \neq \mathsf{Mod}(\mathbf{P}[\mathbf{X}])$. □

The following fact is an alternative formulation of the above definition 3.23.

Fact 3.25 *A functor* $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ *is a semantic functor of a parameterized data type specification* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ *iff:*

1. *there exists a functor* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ *such that for every algebra* $A \in \mathsf{Mod}(\mathbf{X}_\star)$ *there is a tight* $\Sigma_\star$*-monomorphism* $\iota_A : A \to \iota(A)$
2. *For every* $A \in \mathsf{Mod}(\mathbf{X}_\star) : \iota(A) = (\mathsf{F}(A))|_\mu$*, i.e., the following diagram commutes:*

$$
\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[rr, "\mathsf{F}"] \arrow[dr, "\iota"'] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[dl, "|_\mu"] \\
& \mathsf{Mod}(\mathbf{X}_-) &
\end{tikzcd}
$$

(Although the ι here is not the same as in definition 3.23, its role is essentially the same and there is no real danger in confusing the two.) The requirement of ι_A being a monomorphism implies that ι_A must be injective [15]. The tightness requirement ensures that $\iota_A(\star^A) = \mu(\star)^{\mathsf{F}(A)|_\mu}$, i.e., the carrier $s^A = \star_s^A$ is injectively embedded into the carrier $s^{\mathsf{F}(A)}$ as the subset $\mu(\star_s)^{\mathsf{F}(A)}$. Together, the requirements mean that A is a (tight) subalgebra of $\mathsf{F}(A)|_\mu$ and the carrier of this subobject corresponds bijectively in $\mathsf{F}(A)$ to $\mu(\star_s)^{\mathsf{F}(A)}$ – thus ensuring protection of the parameter algebra. This is a generalization of the requirement that F has to be a persistent functor. The classical case of persistency is obtained as the special case when $\mu(\star_s) = \star_s$, for all s.

Example 3.26 *For the specification of stacks from example 3.17 we may, for instance, define the following semantic functor* $\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star)$*:*

- *objects: a given* $A \in \mathsf{Mod}(\mathbf{El}_\star)$ *is mapped to* $\mathsf{F}(A) \in \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star)$ *given by:*
  - $El^{\mathsf{F}(A)} = El^A \cup \{\bot\}$*, where* $\bot \notin El^A$*, and* $Stack^{\mathsf{F}(A)} = (El^A)^*$ *– finite strings of elements from* A
  - $empty^{\mathsf{F}(A)} = \varepsilon$ *– the empty string*
  - $push^{\mathsf{F}(A)}(x, s) = \begin{cases} s & if\ x = \bot \\ xs & otherwise \end{cases}$
  - $pop^{\mathsf{F}(A)}(xs) = s$ *and* $pop^{\mathsf{F}(A)}(\varepsilon) = \varepsilon$
  - $top^{\mathsf{F}(A)}(xs) = x$ *and* $top^{\mathsf{F}(A)}(\varepsilon) = \bot$

– $ok^{\mathsf{F}(A)} = El^A$, $\star_{El}^{\mathsf{F}(A)} = El^{\mathsf{F}(A)}$ *and* $\star_{Stack}^{\mathsf{F}(A)} = Stack^{\mathsf{F}(A)}$.

- *morphisms:* $h : A \to B$ *is mapped to* $\mathsf{F}(h) : \mathsf{F}(A) \to \mathsf{F}(B)$ *given by:*
  – *for* $x \in El^{\mathsf{F}(A)} : \mathsf{F}(h)(x) = \begin{cases} h(x) & \text{if } x \neq \perp \\ \perp & \text{otherwise} \end{cases}$
  – *for* $s \in Stack^{\mathsf{F}(A)}$: $\mathsf{F}(h)(s)$ *is the pointwise application of* $\mathsf{F}(h)(x)$ *to successive* x*'s in* s.

Of course, one might attempt a more specific error treatment but we are here merely illustrating the basic idea. We check that F *satisfies definition 3.23 by verifying the conditions of fact 3.25.*

The functor $\iota : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{El}_-)$ *is given by* $\iota(A) = \mathsf{F}(A)|_\mu$*, which obviously makes the diagram in point 2. (fact 3.25) commute. For each* $A \in \mathsf{Mod}(\mathbf{El}_\star)$*,* ι_A *is the embedding of* A *into* $\mathsf{F}(A)|_\mu$ *(*$\forall x \in |A| : \iota_A(x) = x$*), i.e., it is monomorphism. It is tight (mainly) because we have that* $\iota_A(\star^A) = El^A = (ok^{\mathsf{F}(A)})|_\mu = \mu(\star)^{\mathsf{F}(A)|_\mu}$*. (To obtain this last equality it is essential that the target is required merely to be a model of* $\mathbf{EL}_-$ *and not of* $\mathbf{El}_\star$*. Verification of tightness for other operations is easy.) Thus, although the reduct* $\mathsf{F}(A)|_\mu$ *contains the additional element* $\perp$ *in its carrier, this element does not interfere with the requirement of* A *being a tight subalgebra of this reduct.*

Notice that, thanks to the local guards $x \prec ok \Rightarrow ...$ in axioms 2. and 3. from $\mathbf{Stack}[\mathbf{El}]_\star$, we can actually obtain extension of the carrier with a new *El*ement $\perp$, and yet ignore it when *push*ing on stacks. The above functor illustrates just one possibility of the functor semantics from definition 3.23. There are, of course, more.

A functor sending each $\mathbf{El}_\star$ algebra A to the standard stack algebra where *top*(*empty*) returns the empty set would be another possibility. This would be, in fact, the solution analogous to the free-persistent functor semantics with partial algebras (the local guards $x \prec ok$ could then be dropped in axioms 2. and 3. in $\mathbf{Stack}[\mathbf{El}]_\star$). It is known that partial algebras admit more free persistent functors than the total algebras do, and we can capture this extensions since undefinedness of partial algebra terms can be modeled using empty set (see [7]).

But our extension is much more powerful since, in general, it will admit free functor semantics whenever $\mathbf{P}[\mathbf{X}]_\star$ does not force any new identifications of $\mathbf{X}_\star$ elements (which aren't already forced by $\mathbf{X}_\star$). The example illustrates the possibility of extending the carrier of parameter algebra. If our specifications are deterministic (or, more generally, allow free extensions), we can always choose the free functor semantics (as long as no new identifications are implied). Such a functor can be chosen as the semantics of the specification from example 3.16. This cannot be done with partial algebras alone as long as one uses the classical definition of persistent functor.

Of course, the definition 3.15 of the syntax of PDT does not guarantee the existence of a semantic functor. As in the case of persistency, we would have to show that the parameterized theory $\mathbf{P}[\mathbf{X}]_\star$ is a conservative extension of the parameter $\mathbf{X}_-$, and this problem is undecidable. It is a possible topic for further work to identify syntactic restrictions on the specifications ensuring the existence of a semantic functor.

4 Actual parameter passing

Since we specify PDTs, it might seem that passing an actual parameter amounts merely to applying a semantic functor to the actual (parameter) algebra in order to obtain a new algebra. This, however, is only what happens at the level of programs implementing our specifications. Remaining still at the specification level, we want to allow passing other specifications as actual parameters to a specification of parameterized data type – such an operation should yield a specification of a *new* PDT. That is, instantiating the formal parameter of a PDT by a specification allows one to reuse the PDTs. This will be addressed mostly by the syntactic considerations below and in subsection 4.1. In subsection 4.2 we will show that instantiation at the level of specifications can be reflected at the level of semantic functors. In fact, given a semantic functor F for a PDT

$(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ and an instantiation of the formal parameter $\mathbf{X}_\star$ by an actual parameter $\mathbf{Y}_\star$, we can actually construct a semantic functor for the resulting specification in a canonical way. As in the classical approach to PDTs based on free-persistent functor semantics, this implies the possibility of reusing not only specifications of PDTs but also their actual implementations.

We start with an example of the intended construction. We use the specification of stacks parameterized by elements from example 3.17. As the actual parameter we want to pass the specification of natural numbers from example 3.12. (Recall that, given a standard specification **Nat**, we can obtain the desired form $\mathbf{Nat}_\star$ by applying the syntactic operation $\__\star$ from definition 3.13.)

Let the actual parameter passing morphism – a specification morphism $\nu : \mathbf{El}_\star \to \mathbf{Nat}_\star$ – be given by $\nu(El) = Nat$ and $\nu(\star_{El}) = \star_{Nat}$. The result of instantiation is the following specification

$$
\begin{array}{lrrcl}
\multicolumn{5}{l}{\textbf{spec Stack[Nat]}_\star =} \\
\mathbf{S}: & \multicolumn{4}{l}{Stack, Nat} \\
\Omega: & empty: & & \to & Stack \\
 & top: & Stack & \to & Nat \\
 & pop: & Stack & \to & Stack \\
 & push: & Nat \times Stack & \to & Stack \\
 & zero: & & \to & Nat \\
 & succ: & Nat & \to & Nat \\
S^\star: & \star_{Nat}: & \multicolumn{3}{l}{\to Nat} \\
 & \star_{Stack}: & \multicolumn{3}{l}{\to Stack} \\
 & ok: & \multicolumn{3}{l}{\to Nat}
\end{array}
$$

$$
\begin{array}{llcl}
\Phi: & 1. & & zero \doteq zero \\
 & 2.\ \ x \prec ok & \Rightarrow & succ(x) \doteq succ(x) \\
 & 3.\ \ x \prec ok & \Rightarrow & pred(succ(x)) \doteq x \\
 & 4. & & empty \doteq empty \\
 & 5.\ \ x \prec ok & \Rightarrow & top(push(x,s)) \doteq x \\
 & 6.\ \ x \prec ok & \Rightarrow & pop(push(x,s)) \doteq s \\
\Gamma: & 7. & & x \prec \star_{Nat} \\
 & 8. & & s \prec \star_{Stack}
\end{array}
$$

The first three axioms come from $\mathbf{Nat}_\star$. The thing to observe is that the guards $x \prec \star_{Nat}$ in axioms 2. and 3. from $\mathbf{Nat}_\star$ have changed to $x \prec ok$. This happens because the parameterization morphism was defined by $\mu(\star_{El}) = ok$, and $\delta(\star_{El}) = ok$ prescribed **restricting (local) guards**. The above specification is obtained as a pushout (in the category of multialgebraic specifications $\mathbf{Th}_{\mathcal{MA}}$ – see section 2) of μ and ν:

$$
\begin{tikzcd}
\mathbf{El}_- \arrow[r, "\mu"] \arrow[d, "\nu"'] & \mathbf{Stack}[\mathbf{El}]_\star \arrow[d, "\nu'"] \\
\mathbf{Nat}_- \arrow[r, "\mu'"'] & \mathbf{Stack}[\mathbf{Nat}]_\star
\end{tikzcd}
\tag{3}
$$

Notice that the formal and actual parameter specifications in this diagram are weakened (to $\mathbf{El}_-$, resp. $\mathbf{Nat}_-$) by removing their global guards. The intention of this weakening is to remove the translation $x \prec ok$ of the global guard $x \prec \star_{Nat}$ along μ' from the resulting specification – we want to keep this global guard there, and not the stronger (but not intended) $x \prec \mu'(\star_{Nat}) = ok$. The global guard $x \prec \star_{Nat}$ enters the result along ν' from $\mathbf{Stack}[\mathbf{El}]_\star$.

The result is a (new) PDT $(\mu', \mathbf{Nat}_\star, \mathbf{Stack}[\mathbf{Nat}]_\star, \delta')$, where μ' and δ' are identities except for:

- $\mu'(\star_{Nat}) = ok$

- $\delta'(\star_{Nat}) = ok$.

The latter reflects **restricting (local) guards** in axioms from $\mathbf{Nat}_-$ (2., 3.) according to μ.

4.1 Syntax of actual parameter passing

In order to ensure the existence of the pushout as illustrated in diagram (3) above, we have to ensure that the involved morphisms μ and ν from $\mathbf{El}_-$ are actually specification morphisms. The former is so by fact 3.18.2. As to the latter, we have to take into account a more general situation than in diagram (3), namely, the possibility that the actual parameter passing morphism is not surjective on the sorts, i.e., the actual parameter contains sorts which are not in the image of ν.

The global guards of sorts s' which are in the image of ν should be dropped (and not translated by μ'!). But if $\mathbf{Y}_\star$ contains a sort s which is not in the image of ν, its global guard $x \prec \star_s$ should be preserved in $\mathbf{P}[\mathbf{Y}]_\star$, while the local ones left unchanged (since parameterization does not affect these sorts at all). The treatment of global guards in these two cases calls for the definition 4.1 of weakening a specification $\mathbf{Y}_\star$ not to $\mathbf{Y}_-$ but only to $\mathbf{Y}_{\nu(-)}$ by removing only the global guards which are in the image of a given (signature) morphism ν.

Definition 4.1 *Given signatures $\Sigma_\star = (\mathbf{S}, \Omega, \star)$ and $\Sigma' = (\mathbf{S}', \Omega', \star')$, guarded specifications $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, $\mathbf{SP}'_\star = (\Sigma'_\star, \Phi'_\star, \Gamma_{\Sigma'})$ and a signature morphism $\nu : \Sigma_\star \to \Sigma'_\star$, the* weakening of $\mathbf{SP}'_\star$ along ν *is the specification $\mathbf{SP}'_{\nu(-)} = (\Sigma'_\star, \Phi'_\star, \Gamma_{\Sigma'} \setminus \{x \prec \nu(\star_s) : s \in S\})$.*

This weakening removes only these global guards from the target specification which are in the image of global guards from the source specification. In particular, if $\nu(\star_s) \neq \star_{\nu(s)}$, the global guard $x \prec \star_{\nu(s)}$ will not be removed, but $x \prec \nu(\star_s)$ will be. This definition will be used only in conjunction with the next one.

Definition 4.2 *An* actual parameter passing *is a specification morphism: $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ satisfying $\nu(\star_s) = \star_{\nu(s)}$, for all $s \in \Sigma(\mathbf{X}_-)$.*

The intuition behind the extra requirement $\nu(\star_s) = \star_{\nu(s)}$ should be clear: $\star_s$ in the formal parameter stands for all elements of the carrier of sort s and $\star_{\nu(s)}$ does the same with respect to the elements of the carrier $\nu(s)$ in the actual parameter. Thus ν should identify the two – the restrictions induced on and expressed using $\star_s$ should now be transferred to $\star_{\nu(s)}$.

We register the following simple fact to be of relevance for defining semantics of instantiation and composition.

Lemma 4.3 *Given specifications $\mathbf{X}_\star$ and $\mathbf{Y}_\star$, if $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ is an actual parameter passing, then $\nu : \mathbf{X}_\star \to \mathbf{Y}_\star$ is a specification morphism.*

Proof. Let $\mathbf{X}_- = (\Sigma, \Phi_\star)$, $\mathbf{Y}_{\nu(-)} = (\Sigma', \Phi'_\star, \Gamma'_-)$, and $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ be an actual parameter passing. Suppose that $A \in \mathsf{Mod}(\mathbf{Y}_\star)$. Then

1. First, $A \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ since $\mathbf{Y}_{\nu(-)} \subseteq \mathbf{Y}_\star$. Then, since $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ is a specification morphism we get that $A|_\nu \in \mathsf{Mod}(\mathbf{X}_-)$, so $A|_\nu \models \Phi_\star$.
2. Since $A \in \mathsf{Mod}(\mathbf{Y}_\star)$, we have that $A \models x \prec \star_{s'}$ for all $s' \in \Sigma'$, in particular, $A \models x \prec \star_{\nu(s)}$ for all $s \in \Sigma$. Since $\nu(\star_s) = \star_{\nu(s)}$, we obtain $A|_\nu \models x \prec \star_s$ for all $s \in \Sigma$, i.e., $A|_\nu \models \Gamma_\Sigma$.

1. and 2. mean that $A|_\nu \in \mathsf{Mod}(\mathbf{X}_\star)$, and since A was arbitrary, the claim follows. □

Definition 4.4 *Given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and an actual parameter passing $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$, the result of instantiation (the actual parameter passing) is a specification $\mathbf{P}[\mathbf{Y}]_\star$ obtained by* *pushout (in $\mathbf{Th}_{\mathcal{MA}}$) of ν and μ:*

$$\begin{tikzcd}
\mathbf{X}_{-} \arrow[r, "\mu", "\delta"'] \arrow[d, "\nu"'] & \mathbf{P}[\mathbf{X}]_{\star} \arrow[d, "\nu'"] \\
\mathbf{Y}_{\nu(-)} \arrow[r, "\mu'", "\delta'"'] & \mathbf{P}[\mathbf{Y}]_{\star}
\end{tikzcd}$$

Now, it is not clear that the result will be a PDT. In fact, since pushout is defined only up to isomorphism, this need not be the case. To ensure that it is, we will make a canonical choice of the pushout object.

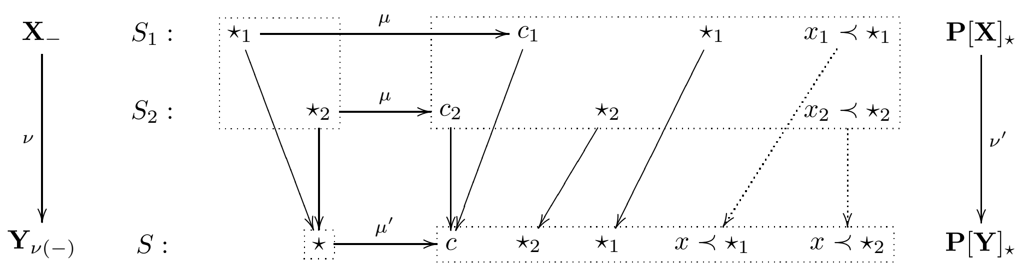
4.1.1 The canonical pushout object

The examples and technicalities in this subsection culminate in lemma 4.8 and proposition 4.9. Impatient reader may only register these facts and continue reading at section 4.2.

In the definition 3.15 of PDT we demand that the parameterization morphism be an inclusion on the reduced signature of the (formal)parameter. Thus μ' has to be the identity on $\Sigma_{-}(\mathbf{Y}_{\nu(-)})$, so the names in the pushout object $\mathbf{P}[\mathbf{Y}]_{\star}$ should be chosen appropriately.

A more intricate question concerns the choice of the sets $\bigstar$ and $C^{\star}$ in the resulting specification. Since morphisms in $\mathbf{Sign}_{\star}$ are the same as in $\mathbf{Sign}$, we may have two different signatures $\Sigma_{\star} \neq \Sigma'_{\star}$, with $\underline{\Sigma_{\star}} = \Sigma = \underline{\Sigma'_{\star}}$, which differ merely by the fact that a constant c in $\Sigma_{\star}$ belongs to $C^{\star}$ while in $\Sigma'_{\star}$ to $\bigstar'$. We want the resulting specification to possess sort constant $\star_s$ for each sort symbol s (assuming that the parameterized specification and the actual parameter do), and also to be fully guarded (assuming that the parameterized specification and the actual parameter are). The idea here is to include among the sort constants $\bigstar_{\mathbf{P}[\mathbf{Y}]_{\star}}$ of the resulting specification all the (sub)sort constants c, such that the specification contains also the respective global guard $x \prec c$. We illustrate it by the following examples.

Example 4.5 *Let $\mathbf{X}_{\star}$ contain two sorts S_1, S_2 and μ send the respective $\mu(\star_i) = c_i$ in $\mathbf{P}[\mathbf{X}]_{\star}$. Let ν identify these two sorts, i.e., $\nu(S_1) = \nu(S_2) = S$. The resulting pushout $\mathbf{P}[\mathbf{Y}]_{\star}$ is shown in the rightmost bottom corner:*

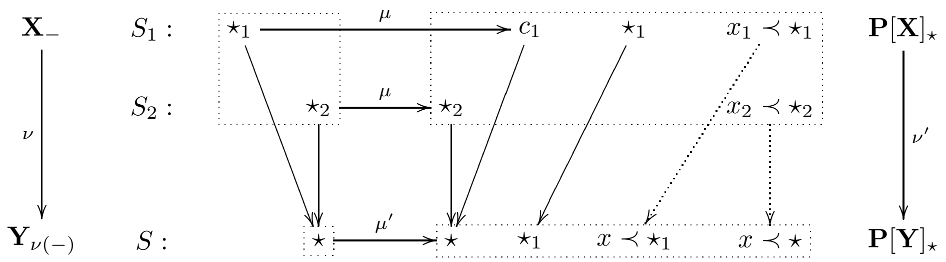


The resulting specification $\mathbf{P}[\mathbf{Y}]_{\star}$ contains two constants $\star_1$ and $\star_2$, both of the same sort S and it also contains both global guards $x \prec \star_1$ and $x \prec \star_2$ originating from the global guards in $\mathbf{P}[\mathbf{X}]_{\star}$.

The example indicates the reason for allowing the set $\bigstar$ to contain more than one constant $\star$ of each sort. Of course, trivial and automatic manipulation may be performed to remove such redundant duplicates, but its result, although in a strong sense equivalent (due to the global guards, the respective model categories are not only equivalent but contain "essentially" the same objects), won't be isomorphic to the pushout specification (due to the difference in the signatures).

Example 4.6 *Let $\mathbf{X}_{\star}$ contain again two sorts S_1, S_2 which are identified by ν, i.e., $\nu(S_1) = \nu(S_2) = S$. But now let μ send $\mu(\star_1) = c_1$ while $\mu(\star_2) = \star_2$ in $\mathbf{P}[\mathbf{X}]_{\star}$. The resulting pushout*

$\mathbf{P}[\mathbf{Y}]_\star$ *is shown in the rightmost bottom corner:*



The resulting specification $\mathbf{P}[\mathbf{Y}]_\star$ *contains again two constants* $\star_1$ *and* $\star$*, both of the same sort* S*, with the respective global guards* $x \prec \star_1$ *and* $x \prec \star$ *originating from the global guards in* $\mathbf{P}[\mathbf{X}]_\star$*. The point now is that* ν' *is sending* $\nu'(c_1) = \star$ *or, in other words, that* $\mu'(\star)$ *is actually a sort constant* $\star \in \bigstar_{\mathbf{P}[\mathbf{Y}]_\star}$ *and not merely a subsort constant* $c \in C^\star_{\mathbf{P}[\mathbf{Y}]_\star}$*. This is because* $\mu(\star_2) = \star_2$ *which "overrides" the fact that* $\mu(\star_1) = c_1$*.*

The examples illustrate the motivation for the following definition of the canonical choice of the pushout specification and, in particular, its signature.

Definition 4.7 *In definition 4.4 we choose the (signature for the) pushout object* $\mathbf{P}[\mathbf{Y}]_\star$ *in the following canonical way.*

Given a pushout signature $\underline{\Sigma(\mathbf{P}[\mathbf{Y}]_\star)}$ *of* μ *and* ν *in* **Sign***, we choose as the names in* $\Sigma(\mathbf{P}[\mathbf{Y}]_\star)$ *all the names coming from* $\overline{\Sigma(\mathbf{Y}_\star)}$ *– the rest of the names are "inherited" from* $\mathbf{P}[\mathbf{X}]_\star$*. (When several subsort constants from* $\mathbf{P}[\mathbf{X}]_\star$ *get identified (like in example 4.5), just choose a fresh name for the resulting subsort constant.)*

As the resulting set of sort constants, $\bigstar_{\mathbf{P}[\mathbf{Y}]_\star}$*, we take the images of all constants* $c \in S^\star_{\mathbf{Y}_\star} \cup S^\star_{\mathbf{P}[\mathbf{X}]_\star}$ *for which we also have a global guard* $x \prec c$*. (These are the images under* ν' *of all* $\bigstar_{\mathbf{P}[\mathbf{X}]_\star}$ *and (the images of) those* $\bigstar_{\mathbf{Y}_\star}$ *which are* not *in the image of* μ *– cf. example 4.6).*

The subsort constants are all the remaining images of subsort constants from $\mathbf{Y}_\star$ *and* $\mathbf{P}[\mathbf{X}]_\star$*, i.e.,*

$$C^\star_{\mathbf{P}[\mathbf{Y}]_\star} = (\nu'[S^\star_{\mathbf{P}[\mathbf{X}]_\star}] \cup \mu'[S^\star_{\mathbf{Y}_\star}]) \setminus \bigstar_{\mathbf{P}[\mathbf{Y}]_\star}.$$

Finally, as the axioms we take the union of the axioms from $\mathbf{Y}_{\nu(-)}$ *(translated along* μ'*) and* $\mathbf{P}[\mathbf{X}]_\star$ *(translated along* ν'*).*

With this choice, we can now state two facts which will be used in the sequel.

Lemma 4.8 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ *with an actual parameter passing* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*, the pushout diagram as in the definition 4.4 and the canonical pushout object from definition 4.7: then* $\nu' : \mathbf{P}[\mathbf{X}]_- \to \mathbf{P}[\mathbf{Y}]_{\nu(-)}$ *is an actual parameter passing.*

Proof. Let $\mathbf{P}[\mathbf{X}]_\star = (\Sigma'_\star, \Phi'_\star, \Gamma_{\Sigma'_\star})$. That $\nu'(\star_{s'}) = \star_{\nu'(s')}$ for all $s' \in \Sigma'_\star$ follows from the pushout properties, since if $\star_{s'} = \mu(\star_s)$, then $\nu'(\star_{s'}) = \nu(\star_s) = \star_{\nu(s)}$ (since ν is an actual parameter passing), and otherwise $\nu'(\star_s) = \star_{\nu'(s')}$. Furthermore, ν' is a specification morphism, i.e., $\mathbf{P}[\mathbf{Y}]_{\nu(-)} \models \mathbf{P}[\mathbf{X}]_-$, since for every formula $\phi_\star \in \mathbf{P}[\mathbf{X}]_- : \nu'(\phi_\star) \in \mathbf{P}[\mathbf{Y}]_{\nu(-)}$, by the canonical pushout construction. □

Proposition 4.9 *With* $\mathbf{P}[\mathbf{Y}]_\star$ *and* μ' *from definition 4.7,* $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star, \mu')$ *is a PDT.*

Proof. All global guards from $\mathbf{Y}_{\nu(-)}$ get included in $\mathbf{P}[\mathbf{Y}]_\star$. Translations along μ (and then ν') of global guards $\Gamma_{\Sigma(\mathbf{X}_\star)}$ originating from $\mathbf{X}_\star$ are "forgotten" (by starting from $\mathbf{X}_-$), but the global guards themselves are passed from $\mathbf{P}[\mathbf{X}]_\star$ along ν'. Thus $\mathbf{P}[\mathbf{Y}]_\star$ is globally guarded (if $\mathbf{Y}_\star$ and $\mathbf{P}[\mathbf{X}]_\star$ are). Also, if $\mathbf{Y}_\star$ and $\mathbf{P}[\mathbf{X}]_\star$ are fully guarded, then so is $\mathbf{P}[\mathbf{Y}]_\star$.

μ' is identity by the choice of the names in $\mathbf{P}[\mathbf{Y}]_\star$, possibly with the exception of some $\star \in \bigstar_{\mathbf{Y}_\star}$, which are in the image of $\bigstar_{\mathbf{X}_\star}$ and are mapped along μ (example 4.5). But these are then mapped to fresh constants, either as by μ, or by the choice of the canonical $\mathbf{P}[\mathbf{Y}]_\star$.

The local guard mapping $\delta' = \mu'$ makes inclusion of the axioms $\mu'(\star_{s'}) \prec \delta'(\star_{s'})$ unnecessary.

Finally, by the canonical pushout construction for all the axioms $\phi_\star \in \mathbf{Y}_{\nu(-)}$ (i.e., except some global guards), we have $\mu'(\phi_\star) \in \mathbf{P}[\mathbf{Y}]_\star$, so that the point 5 of definition 3.15 is satisfied. $\square$

The importance of this fact is that we *always* can see the result of instantiation as a PDT. However, in some cases, it might seem more natural to choose δ' in a more specific way.

Example 4.10 *Assume that $\mu(\star) = c \neq \star = \delta(\star)$ and ν is an actual parameter passing in virtue of being simply an inclusion (translation) of the axioms:*

$$\begin{tikzcd}
\mathbf{X}_- & x \prec \star \ldots \Rightarrow \ldots \arrow[r, "\mu(\star)=c", "\delta(\star)=\star"'] \arrow[d, "\nu(\star)=\star"'] & x \prec \star \ldots \Rightarrow \ldots \arrow[d, "\nu'(\star)=\star"] & \mathbf{P}[\mathbf{X}]_\star \\
\mathbf{Y}_{\nu(-)} & x \prec \star \ldots \Rightarrow \ldots \arrow[r, "\mu'(\star)=c", "\delta'(\star)=?"'] & \begin{array}{c} x \prec \star \ldots \Rightarrow \ldots \\ x \prec c \ldots \Rightarrow \ldots \end{array} & \mathbf{P}[\mathbf{Y}]_\star
\end{tikzcd}$$

Since the PDT allows extension of the carrier ($\mu(\star) = c$) and, at the same time, stipulates the old axioms to remain valid for the new elements ($\delta(\star) = \star$), we migh in this case expect the δ' to do the same. Indeed, in this special case, it may be natural to define δ' corresponding to μ' by

$$\delta'(\star_{s'}) = \begin{cases} \nu'(\delta(\star_s)) & \text{if for some } s \in \Sigma(\mathbf{X}_\star) : \nu(s) = s' \\ \star_{s'} & \text{otherwise} \end{cases}$$

Remember that both specifications on the right ($\mathbf{P}[\mathbf{X}]_\star$ and $\mathbf{P}[\mathbf{Y}]_\star$) have all global guards, i.e., $x \prec \star$. The μ' translation of the axiom yields a weaker guard ($x \prec c \ldots$) than the ν' inclusion of the respective axiom from $\mathbf{P}[\mathbf{X}]_\star$. Indeed, the former is redundant in the presence of the latter – an isomorphic specification would be obtained by just dropping the μ' translation. Thus, in this case, we could safely use the above definition of δ' instead of the general one from proposition 4.9, since pushout is defined up to isomorphism.

However, the above example illustrates only a special case. This definition of δ' would not work in a more general situation.

Example 4.11 *Let $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ be a PDT and $\nu : \mathbf{X}_- \to \mathbf{Y}_-$ be an actual parameter passing (signature inclusion) as shown below (we only write relevant axioms):*

$$\begin{tikzcd}
\mathbf{X}_- & x \prec \star \Rightarrow f(x) \doteq g(x) \arrow[r, "\mu(\star)=c", "\delta(\star)=\star"'] \arrow[d, "\nu(\star)=\star"'] & x \prec \star \Rightarrow f(x) \doteq g(x) \arrow[d, "\nu'(\star)=\star"] & \mathbf{P}[\mathbf{X}]_\star \\
\mathbf{Y}_- & \begin{array}{c} x \prec \star \Rightarrow f(x) \doteq a \\ x \prec \star \Rightarrow a \doteq g(x) \end{array} \arrow[r, "\mu'(\star)=c", "\delta'(\star)=?"'] & \begin{array}{c} x \prec \star \Rightarrow f(x) \doteq g(x) \\ x \prec c \Rightarrow f(x) \doteq a \\ x \prec c \Rightarrow a \doteq g(x) \end{array} & \mathbf{P}[\mathbf{Y}]_\star
\end{tikzcd}$$

If we replaced the local guards $x \prec c \ldots$ in the resulting $\mathbf{P}[\mathbf{Y}]_\star$ by $x \prec \star \ldots$, i.e., applied δ' as defined in the above example 4.10, we would obtain a PDT $\mathbf{P}[\mathbf{Y}]'_\star$ but it would not be a pushout object in $\mathbf{Th}_{\mathcal{MA}}$!

There is, of course, a canonical construction which replaces the local guards $x \prec c \Rightarrow \ldots$ in the specification $\mathbf{P}[\mathbf{Y}]_\star$ resulting from the pushout construction, by $x \prec \star \ldots$ leading to another specification $\mathbf{P}[\mathbf{Y}]'_\star$. There is also an obvious specification morphism from the former to the latter (since, in the presence of global guard $x \prec \star$, we have $(x \prec \star, \overline{a} \Rightarrow \overline{b}) \models (x \prec c, \overline{a} \Rightarrow \overline{b})$, for any (sub)sort constant c). And finally, the specification $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]'_\star, \delta')$, where δ' is as in example 4.10, is a PDT. Thus, we expect that one can obtain more flexibility in passing actual parameters, but the details of that need to be postponed to a future work. For the moment, we are satisfied with the proposition 4.9, and ignore the details and possibility of more specific choices of the PDTs resulting from instantiation.

4.2 Semantics of actual parameter passing

The first aspect of the semantics of instantiation is expressed in proposition 4.9 – it gives a specification of a new parameterized data type. That is, we can reuse PDTs for constructing new PDTs by instantiating the formal parameter.

There is, however, another semantic aspect which will be called "actualization". Given a functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ defining semantics of a parameterized data type specification according to definition 3.23 and an actual parameter passing $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ we want to define the way of transforming $\mathbf{Y}_\star$ algebras into $\mathbf{P}[\mathbf{Y}]_\star$ algebras, i.e., a functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$ which is *induced* by F. In addition, we want this functor to satisfy the conditions corresponding to those imposed on the semantics of parameterized data type specifications (fact 3.25).

Let us return to the example of parameterized specification of stacks from example 3.17, the chosen semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{Stack}[\mathbf{El}]_\star$ from example 3.26 and the actual parameter passing $\nu : \mathbf{El}_- \to \mathbf{Nat}_-$ from the beginning of this section. Let N be the standard $\mathbf{Nat}_\star$ algebra (i.e., the standard $\mathbf{Nat}$ algebra with $\star^N_{Nat}$ being all the natural numbers). The functor F' will simply embed N into $\mathsf{F}'(N)$ and mimic the action of F with respect to constructing the rest of $\mathbf{Stack}[\mathbf{Nat}]_\star$ algebra $\mathsf{F}'(N)$.

The important point is that such an actualization of functor F to a functor F' can be done canonically given a semantic functor F with the corresponding $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$. This corresponds to the classical case of free-persistent functor semantics of parameterization in the presence of amalgamation lemma. Notice, however, that our result is far more general, since we show it for any functor satisfying definition 3.23. Thus we do not require persistency (but allow extension of the carrier) and, furthermore, the extension need not be free, i.e., free functors are only special cases. To show this, we will need the following definition.

Definition 4.12 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$*, the semantic functor* $\mathsf{F} : \mathbf{X}_\star \to \mathbf{P}[\mathbf{X}]_\star$*, with corresponding* ι *and an actual parameter passing morphism* $\nu_- : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*. A functor* $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ *is* induced by ι *iff for all* $Y \in \mathsf{Mod}(\mathbf{Y}_\star)$*:*

1. *there is a tight monomorphism* $\iota'_Y : Y \to \iota'(Y)$*, and*
2. $\iota(Y|_\nu) = \iota'(Y)|_\nu$.

The second point uses overloaded notion of ν which is admissible by lemma 4.3. It means the commutativity of the leftmost square (2.) in the diagram below. The rest of the diagram is referred to in the following definition.

$$\begin{tikzcd}
 & \mathsf{Mod}(\mathbf{X}_\star) \arrow[d, "\iota"] \arrow[rrd, "\mathsf{F}"] & & \\
\mathsf{Mod}(\mathbf{Y}_\star) \arrow[ru, "|_\nu"] \arrow[d, "\iota'"'] \arrow[rrd, "\mathsf{F}'"] \arrow[r, phantom, "2."] & \mathsf{Mod}(\mathbf{X}_-) \arrow[rr, phantom, "1." above] \arrow[rd, phantom, "3."] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[ll, "|_\mu"'] \\
\mathsf{Mod}(\mathbf{Y}_{\nu(-)}) \arrow[ru, "|_\nu"'] & & \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star) \arrow[ll, "|_{\mu'}"] \arrow[ru, "|_{\nu'}"'] &
\end{tikzcd}$$

Notice that the lower triangle of this diagram is actually a more specific case of the general requirement, namely, that ι' is a functor $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_-)$, since $\mathsf{Mod}(\mathbf{Y}_{\nu(-)}) \subseteq \mathsf{Mod}(\mathbf{Y}_-)$. It corresponds to the fact that carrier of any sort s from $\mathbf{Y}_\star$, which is not in the image of ν, is not extended in the models of $\mathbf{P}[\mathbf{Y}]_\star$ (or else, that μ' maps its $\star_s$ and the respective global guard to the same $\star_s$ and global guard in $\mathbf{P}[\mathbf{Y}]_\star$).

Definition 4.13 *Let* F, ι *be as in the previous definition (4.12) and* ι' *be induced by* ι. *The instantiation functor* $\mathsf{F}' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$ *is then defined by:*

- *objects:* $\mathsf{F}'(Y) = \iota'(Y) \oplus_{\iota'(Y)|_\nu} \mathsf{F}(Y|_\nu)$
- *homomorphisms:* $\mathsf{F}'(h) = \iota'(h) \oplus_{\iota'(h)|_\nu} \mathsf{F}(h|_\nu)$

The notation $Y \oplus_X Z$ denotes the amalgamated sum of Y and Z with respect to the common reduct X (cf. section 2.1).

F' is well-defined since, by commutativity of 1. and 2. in the diagram above, $\iota'(Y)|_\nu = (\mathsf{F}(Y|_\nu))|_\mu$ and $\iota'(h)|_\nu = (\mathsf{F}(h|_\nu))|_\mu$. With this definition of F', all the loops in the diagram above commute.

By continuity of the Mod-functor (from $\mathcal{MA}$, section 2), 3. is a pullback diagram (since the corresponding specification was constructed as pushout according to definition 4.4). Hence, by amalgamation property, $\mathsf{F}'(Y)$ is indeed guaranteed to belong to $\mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$.

We thus have the construction of the desired actualization functor provided that we have a functor ι' induced by ι. The following proposition shows that such an induced functor can always be obtained providing also a way to construct it.

Proposition 4.14 *Given a functor* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ *(associated with the semantic functor for the parameterized data type specification) and an actual parameter passing morphism* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$, *there exist a functor* $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ *induced by* ι.

Proof. Let the formal parameter specification be $\mathbf{X}_\star = ((\mathbf{S}, \Omega), \Phi, \Gamma_\mathbf{S})$ and the actual parameter specification $\mathbf{Y}_\star = ((\nu(\mathbf{S}) \cup \mathbf{S}', \nu(\Omega) \cup \Omega'), \nu(\Phi) \cup \Phi', \nu(\Gamma_\mathbf{S}) \cup \Gamma_{\mathbf{S}'})$, (where $\nu(\mathbf{S}) \cap \mathbf{S}' = \nu(\Omega) \cap \Omega' = \nu(\Gamma_\mathbf{S}) \cap \Gamma_{\mathbf{S}'} = \emptyset$). We let $\star$'s to be included in respective Ω's, since they do not require separate treatment here. s/ω range over symbols from $\mathbf{S}/\Omega$ and s'/ω' from $\mathbf{S}'/\Omega'$. The algebra $V = \iota'(Y)$ is constructed by inheriting the sorts and operations which are not in the image of ν directly from Y, while those which are in the image of ν from $\iota(Y|_\nu)$. We define ι' on:

- algebras: for every $Y \in \mathsf{Mod}(\mathbf{Y}_\star)$ we let $\iota'(Y) = V \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ be the following algebra:
  1. sorts:
     (a) for $s \in \mathbf{S} : \nu(s)^V = s^{\iota(Y|_\nu)}$
     (b) for $s' \in \mathbf{S}' : s'^V = s'^Y$
  2. operations:
     (a) for $\omega \in \Omega : \nu(\omega)^V = \omega^{\iota(Y|_\nu)}$,
     (b) for $\omega' \in \Omega' : \omega'(\overline{y})^V = \begin{cases} \omega'(\overline{y})^Y & \textit{if all } \overline{y} \in |Y| \\ \emptyset & \textit{otherwise (if some } \overline{y} \notin |Y|) \end{cases}$
- homomorphisms: $\iota'(h) = h$, i.e., $\iota'(h_{\nu(s)}) = \iota(h_s)$ and $\iota'(h_{s'}) = h_{s'}$.

We let $\iota'(y) = y$ and show that ι' satisfies definition 4.12. This defining equation makes it trivially a monomorphism. It is, indeed, a tight $\Sigma(\mathbf{Y}_\star)$-homomorphism because: for all $\omega \in \Omega$, i.e., $\nu(\omega) \in \nu(\Omega)$, and all $\overline{y} \in |Y| : \nu(\omega)(\overline{y})^V \overset{2a}{=} \omega(\overline{y})^{\iota(Y|_\nu)} = \omega(\overline{y})^{Y|_\nu} = \omega(\overline{y})^Y$, where the middle equality holds since ι is tight homomorphism, and the last one by the definition of reduct. For the remaining operation $\omega' \in \Omega'$ and $\overline{y} \in |Y| : \omega'(\overline{y})^V \overset{2b}{=} \omega'(\overline{y})^Y$. Furthermore, we have that $V|_\nu = \iota(Y|_\nu)$, by construction.

By construction, i.e., by 1b, V satisfies all the global guards in $\Gamma_{\mathbf{S}'}$. By proposition 3.22, it also satisfies all the fully guarded axioms of $\mathbf{Y}_\star$, since ι' is a tight mono. Thus $V \in \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$. □

We have thus shown the possibility of reusing specifications of parameterized data types not only by syntactic instantiation (def. 4.4) but also by providing the above construction of canonical extension of the semantics.

Example 4.15 *In the final example of the whole setting, we use also the possibility offered by multialgebras to model nondeterminism. We specify a generic extension of a deterministic data type with a (binary) nondeterministic choice: we give a (generic) parameterized data type specification and instantiate it for a more specific data type.*

The parameterization morphism μ is an inclusion (i.e., $\mu(\star_{El}) = \star_{El}$), that is, we do not extend carrier:

$$\boxed{\begin{array}{ll} \textbf{spec } \mathbf{El}_\star = & \\ \mathbf{S}: & El \\ \Omega: & \star_{El}: \quad \to El \\ & \\ \Gamma: & 1. x \prec \star_{El} \end{array}} \xrightarrow[\delta(\star_{El}) = \star_{El}]{\mu(\star_{El}) = \star_{El}} \boxed{\begin{array}{lllll} \multicolumn{5}{l}{\textbf{spec } \sqcup[\mathbf{El}]_\star =} \\ \mathbf{S}': & El & & & \\ \Omega': & \sqcup: & El \times El & \to & El \\ & \star_{El}: & & \to & El \\ \Phi': & 1. & & & x \prec x \sqcup y \\ & 2. & & & y \prec x \sqcup y \\ & 3. & z \prec x \sqcup y & \Rightarrow & z \doteq x, z \doteq y \\ \Gamma': & 4. & & & x \prec \star_{El} \end{array}}$$

As the semantic functor we take the free functor $\mathsf{F} : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\sqcup[\mathbf{El}]_\star)$, i.e. for an $\mathbf{El}_\star$ algebra A, $\mathsf{F}(A)$ is given by:

- $|\mathsf{F}(A)| = |A|$
- $x \sqcup^{\mathsf{F}(A)} y = \{x, y\}$

Any actual instantiation can now be associated with an actualization of this functor. Using a specification of natural numbers as actual parameter we get the specification of natural numbers with binary choice as the result. The corresponding semantic functor (induced by F and the obvious parameter passing $\nu(El) = Nat$), $\mathsf{F}' : \mathsf{Mod}(\mathbf{Nat}_\star) \to \mathsf{Mod}(\sqcup[\mathbf{Nat}]_\star)$, will embed an arbitrary $\mathbf{Nat}_\star$ algebra A into $\mathsf{F}'(A)$ by

- $|\mathsf{F}(A)| = |A|$
- $x \sqcup^{\mathsf{F}(A)} y = \{x, y\}$
- $\omega^{\mathsf{F}(A)} = \omega^A$ *– for all operation symbols $\omega \in \Sigma(\mathbf{Nat}_\star)$.*

One should keep in mind that although syntax and semantics of instantiated specification are obtained from the parameterized specification itself, the two represent specifications of two distinct – and, as a mater of fact, unrelated – (parameterized) data types.

One could probably think of a more general means of specifying instantiation mechanisms at the algebra (program) level, that is, mechanisms of taking a PDT and matching an actual parameter *algebra* in order to obtain a new data type (and not merely, as we have now, a PDT which can be applied to actual parameter algebras coming only from the model class of the parameter specification). The actualization mechanism described above would be a special case of such a more general instantiation in that the "matching" of actual parameter algebras here is expressed by the reduct functor from $\mathsf{Mod}(\mathbf{Y}_\star)$ to $\mathsf{Mod}(\mathbf{X}_\star)$. This would require a closer look at the possibilities of describing the semantic functors and we have to leave such considerations for future work.

5 Composition and refinement

We will now review various ways of composing specifications of parameterized data types. We will discuss the classical vertical and horizontal composition, showing the counterparts of the standard compositionality theorems. The main difference will concern the fact that, in general, stepwise application of constructions will not yield the same result as a direct construction along the respective composition, but a refinement of the latter. Subsections 5.1 and 5.2 discuss vertical, and 5.3 and 5.4 horizontal composition. Section 6 will summarize the concept of refinement which emerges from this section.

We recall that, given a parameter passing diagram (like 1. below in Figure 1), by proposition 4.9, $\mu' : \mathbf{Y}_{\nu(-)} \to \mathbf{P}[\mathbf{Y}]_\star$ is a parameterization morphism, and hence, in particular (by fact 3.18), $\mu' : \mathbf{Y}_- \to \mathbf{P}[\mathbf{Y}]_\star$ is a specification morphism.

5.1 Vertical composition

Given two actual prameter passing morphisms $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ and $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$, (as indicated in the diagrams 1. and 2. in Figure 1), we would like to compose them vertically, i.e., we want to show that also $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ is an actual parameter passing.

$$
\begin{tikzcd}
\mathbf{X}_- \arrow[r, "\mu"] \arrow[d, "\nu"'] \arrow[dr, phantom, "1."] \arrow[dddd, bend right, "\nu;\rho"'] & \mathbf{P}[\mathbf{X}]_\star \arrow[d, "\nu'"] \arrow[dddd, bend left, "(\nu;\rho)'"] \\
\mathbf{Y}_{\nu(-)} \arrow[r, "\mu'"] & \mathbf{P}[\mathbf{Y}]_\star \\
\mathbf{Y}_- \arrow[r, "\mu'"] \arrow[d, "\rho"'] \arrow[dr, phantom, "2."] & \mathbf{P}[\mathbf{Y}]_\star \arrow[d, "\rho'"] \\
\mathbf{Z}_{\rho(-)} \arrow[r, "\mu''"] & \mathbf{P}[\mathbf{Z}]_\star \\
\mathbf{Z}_{(\nu;\rho)(-)} \arrow[r, "\mu'''"] & \mathbf{P}[\mathbf{Z}]'_\star
\end{tikzcd}
$$

Figure 1:

The notation from this figure will be used throughout this and next subsection (5.1, 5.2).

In general, the specifications $\mathbf{Z}_{\rho(-)}$ and $\mathbf{Z}_{(\nu;\rho)(-)}$ need not be the same – the latter may have more global guards than the former.

Fact 5.1 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star)$ *and actual parameter passing* ν *and* ρ *as in the Figure 1:*

1. $\mathbf{Z}_{(\nu;\rho)(-)} \models \mathbf{Z}_{\rho(-)}$.
2. *If* ν *is surjective on the sorts, then* $\mathbf{Z}_{\rho(-)} \models \mathbf{Z}_{(\nu;\rho)(-)}$.
3. *If* ν *is surjective on the sorts, then* $\mathbf{Z}_{(\nu;\rho)(-)} \simeq \mathbf{Z}_{\rho(-)}$.

Proof. Direct from definition 4.1. Obviously, both specifications have isomorphic signatures (so, for simplicity, we assume that the are equal). Also, all the axioms except, possibly, some global guards, are involved in both pushout constructions and will be satisfied by both specifications. The only difference may concern absence in $\mathbf{Z}_{\rho(-)}$ of some global which are present in $\mathbf{Z}_{(\nu;\rho)(-)}$.

1. All sorts which are in the image of $(\nu;\rho)$ are also in the image of ρ, so the global guards dropped in $\mathbf{Z}_{(\nu;\rho)(-)}$ are also dropped in $\mathbf{Z}_{\rho(-)}$.
2. If ν is surjective on the sorts then if a sort is in the image of ρ it will also be in the image of $(\nu;\rho)$. Hence all global guards from $\mathbf{Z}_{(\nu;\rho)(-)}$ will also be present in $\mathbf{Z}_{\rho(-)}$.
3. If ν is surjective on the sorts, then the isomorphism follows from the two points above.

□

Notice that, in points 2. and 3., surjectivity of ν on sorts is sufficient but not necessary condition. It is sufficient and necessary that for any sort $s \in \Sigma(\mathbf{Y})$ which is not in the image of ν, there is a sort $s' \in \Sigma(\mathbf{Y})$ which is in the image of ν and such that $\rho(s) = \rho(s')$.

Proposition 5.2 *If* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ *and* $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$ *are actual parameter passing morphisms, then so is* $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ *(see the diagram in Figure 1).*

Proof. We have both $\nu(\star_s) = \star_{\nu(s)}$ and $\rho(\star_{s'}) = \star_{\rho(s')}$ for all sort symbols $s \in \Sigma(\mathbf{X})$ and $s' \in \Sigma(\mathbf{Y})$, and thus $(\nu;\rho)(\star_s) = \star_{(\nu;\rho)(s)}$. We show that $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ is a specification morphism, i.e., $\mathbf{Z}_{(\nu;\rho)(-)} \models (\nu;\rho)(\mathbf{X}_-)$.

$$\begin{array}{rl} \text{fact 5.1}: & \mathbf{Z}_{(\nu;\rho)(-)} \models \mathbf{Z}_{\rho(-)} \\ \rho \text{ is a specification morphism}: & \mathbf{Z}_{\rho(-)} \models \rho(\mathbf{Y}_-) \\ \Rightarrow & \mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\mathbf{Y}_-) \end{array}$$

Now, the axioms of $\mathbf{Y}_{\nu(-)} = (\Phi, \Gamma')$, where Γ' is the subset of global guards from $\mathbf{Y}_\star$ which (whose sort symbols) are not in the image of ν. To complete the proof we have to show that $\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\Gamma')$. But this follows directly from definition 4.1. For any global guard $\gamma \in \Gamma'$ is *not* in the image of ν and hence it will *not* be in the image of $\nu;\rho$. Consequently, if $\gamma \in \Gamma'$ then $\rho(\gamma) \in \mathbf{Z}_{(\nu;\rho)(-)}$ (though not necessarily $\rho(\gamma) \in \mathbf{Z}_{\rho(-)}$!!).

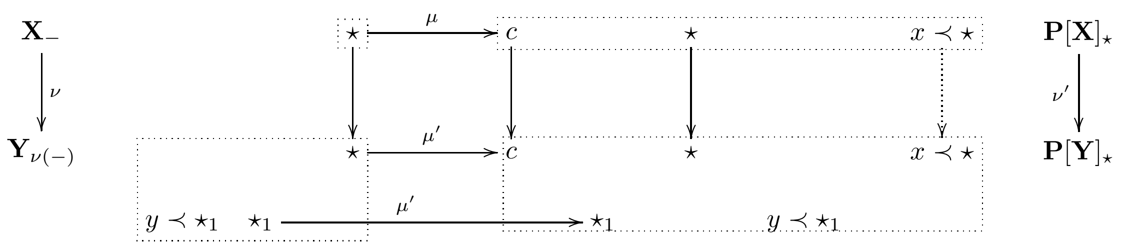
Thus $\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\Gamma')$ which together with (5.1) yields

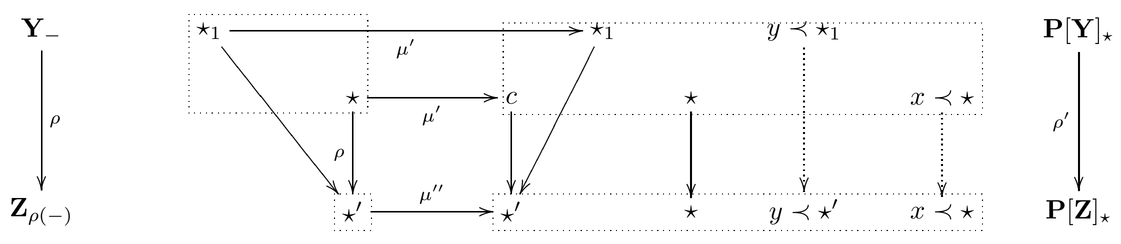
$$\mathbf{Z}_{(\nu;\rho)(-)} \models \rho(\mathbf{Y}_{\nu(-)}). \tag{4}$$

In diagram 1. ν is a parameter passing, so $\mathbf{Y}_{\nu(-)} \models \nu(\mathbf{X}_-)$ which implies $\rho(\mathbf{Y}_{\nu(-)}) \models (\nu;\rho)(\mathbf{X}_-)$. This, together with (4) give the conclusion: $\mathbf{Z}_{(\nu;\rho)(-)} \models (\nu;\rho)(\mathbf{X}_-))$. $\square$

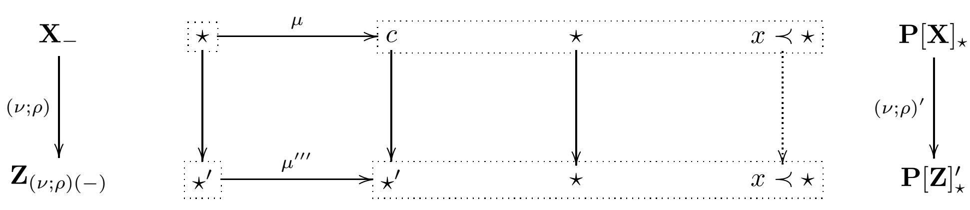
In general, the specifications $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ may be different. In the classical case, this is merely a consequence of their definition by pushout (which is unique only up to isomorphism). In our case, however, the difference may be more significant, since we also may drop and/or add some global guards on the way. As in fact 5.1, the only difference may concern the presence/absence of global guards (since all other axioms are involved in the pushout construction), so these are the only axioms we mention in the following example.

Example 5.3 *Consider first two instantiations* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ *and* $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$*. (Two lines in* $\mathbf{Y}_{\nu(-)}$*,* $\mathbf{Y}_-$*, etc. represent two distinct sorts which are identified by the second instantiation* ρ*.)*





And now a direct instantiation along $\nu;\rho$*:*



The significant difference consists in that $\mathbf{P}[\mathbf{Z}]_\star$ *has the global guard for* $\mu''(\star)$*, namely* $y \prec \star'$ *originating from* $\mathbf{P}[\mathbf{Y}]_\star$*. Thus here* $\bigstar = \{\star, \star'\}$ *and* $C^\star = \emptyset$*. In* $\mathbf{P}[\mathbf{Z}]'_\star$*, on the other hand, this guard is not present, so here* $\bigstar' = \{\star\}$*, while* $C^{\star\prime} = \{\star'\}$*.*

Thus, the PDT $(\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star)$ *would forbid extending the carrier of* $\star'$*, while* $(\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star)$ *would not.*

So, in general, $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ are not isomorphic. We have the following fact.

Fact 5.4 *With the notation from Figure 1 and example 5.3:*

1. $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$.
2. *if* $\mathbf{P}[\mathbf{Z}]'_\star \not\models \mathbf{P}[\mathbf{Z}]_\star$*, then it is only because for some sort constant(s)* $c : \mathbf{P}[\mathbf{Z}]_\star \models x \prec c$ *and* $\mathbf{P}[\mathbf{Z}]'_\star \not\models x \prec c$.

Proof. The signatures of both specifications will be isomorphic, so we assume that they are identical. All axioms except global guards are involved in the pushout constructions, so their presence (or satisfaction) follows from the standard isomorphism of pushout objects. The difference may concern only some constants which are in $\bigstar$ but not in $\bigstar'$ (only in $C^{\star\prime}$, as in the example 5.3). This justifies point 2. For point 1. we show that if $\star \in \bigstar'$ then $\star \in \bigstar$, that is if $\mathbf{P}[\mathbf{Z}]'_\star \models x \prec \star$ then $\mathbf{P}[\mathbf{Z}]_\star \models x \prec \star$, which will yield the conclusion.

This follows trivially. Any global guard $x \prec \star$ in $\mathbf{P}[\mathbf{Z}]'_\star$ is an image of a respective global guard either from $\mathbf{Z}_{(\nu;\rho)(-)}$ or from $\mathbf{P}[\mathbf{X}]_\star$. In the latter case, it will also be present in $\mathbf{P}[\mathbf{Y}]_\star$ and hence also in $\mathbf{P}[\mathbf{Z}]_\star$.

In the former case, if this guard is also in $\mathbf{Z}_{\rho(-)}$ it will be present in $\mathbf{P}[\mathbf{Z}]_\star$. If it does not belong to $\mathbf{Z}_{\rho(-)}$, this means that it (its sort) is in the image of ρ (and therefore was dropped). But then, its ρ pre-image must be in $\mathbf{Y}_\star$, that is, must be present in $\mathbf{P}[\mathbf{Y}]_\star$. But then it is also present in $\mathbf{P}[\mathbf{Z}]_\star$ as the result of pushout construction. □

The fact that stepwise instantiation (of $\mathbf{X}_\star$ by $\mathbf{Y}_\star$ and then by $\mathbf{Z}_\star$ leading to $\mathbf{P}[\mathbf{Z}]_\star$) yields a different result than the direct instantiation (of $\mathbf{X}_\star$ by $\mathbf{Z}_\star$ leading to $\mathbf{P}[\mathbf{Z}]'_\star$) may look like a severe weakness of our setting. After all, equality of these two indicates the desirable compositionality which would be expected by anybody familiar with the traditional, pushout based theory of parameterized specifications.

However, we are not developing a theory of parameterized specifications but of specification of parameterized data types. This means, we are interested in constructions allowing us to obtain new data types (algebras) from others. In this setting, performing different series of constructions or, as in the case of vertical composition, performing constructions in different ways, may be expected to yield different results.

Our point is that stepwise instantiation, first along ν and then along ρ represents a slightly different construction than the direct instantiation along $\eta = \nu; \rho$. In fact, we suggest to think of the former as a refinement of the latter. The latter is a one step construction along η. In this sense, splitting this construction in two steps, first along ν and then ρ, is a more detailed, refined construction which may introduce new aspects. We certainly want the result of this refined construction to be "compatible" with the results prescribed by the more rough, one step construction. This is the meaning of one construction refining another which corresponds to the classical concept of refinement by model class inclusion. This is indicated by the fact 5.4.1 and we now proceed to illustrate the semantic aspect of this refinement.

5.2 Vertical composition – semantics

As noted in section 4.2, we can view the semantics of instantiation from two angles: on the one hand, as a new PDT with a class of its semantic functors and, on the other hand, as an actualization: a functor for the resulting PDT induced by a particular functor for the instantiated PDT. We now apply this distinction in the discussion of the semantics of vertical composition.

5.2.1 Vertical composition as a refinement of PDT

We postpone the general definition of refinement to section 6 and for the moment take it intuitively to mean: a PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ is a *refinement* of a PDT $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$, $\mathbf{P}' \rightsquigarrow \mathbf{P}$, if any semantic functor for $\mathbf{P}$ can be used for obtaining a semantic functor for $\mathbf{P}'$.

A trivial, though by no means only, example of such a refinement is when $\mathbf{P}[\mathbf{X}]'_\star \rightsquigarrow \mathbf{P}[\mathbf{X}]_\star$, i.e., $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \supseteq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, while other components are equal. This is, in fact, the case with the results of vertical composition. If we view $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ as two independent PDTs (i.e., "forget" that they both originate from instantiation of the same PDT), we see that, by fact 5.4, $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$, i.e., we have an inclusion (functor) $\mathsf{i} : \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$. Thus any semantic functor F for $\mathbf{P} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \delta'')$ gives a semantic functor for $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$, simply by composing $\mathsf{F};\mathsf{i}$. The other components of both PDTs are (essentially) the same, and so we get

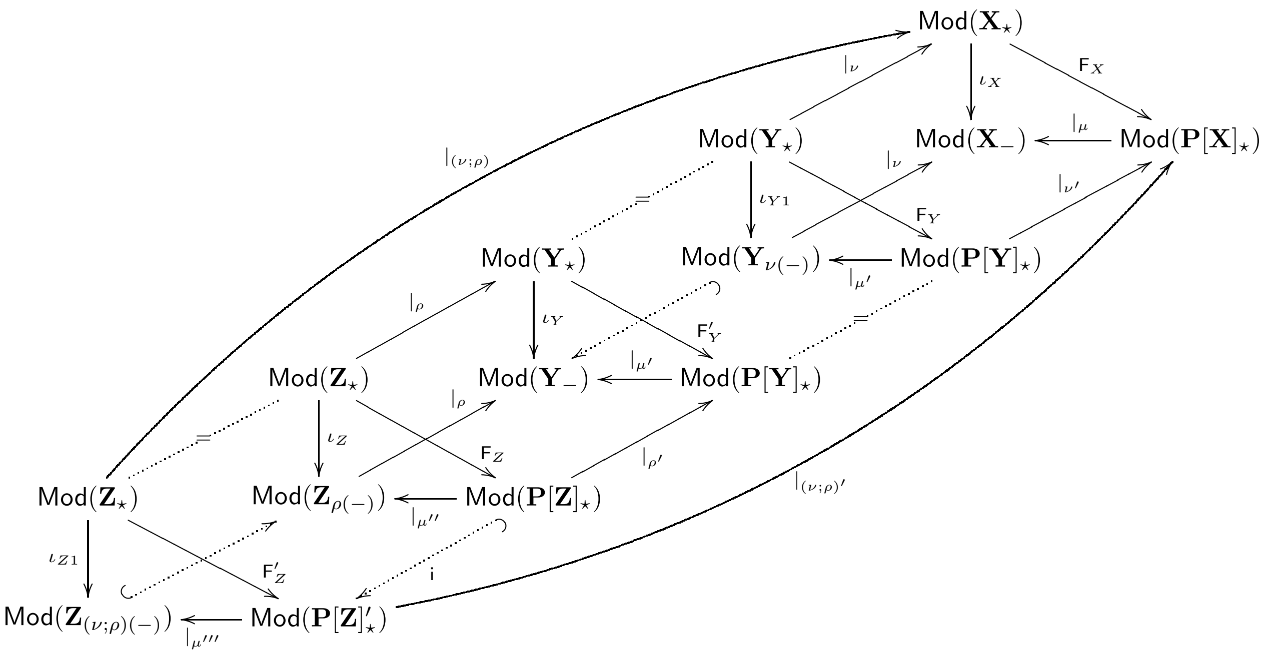
Fact 5.5 *Given* $\mathbf{P} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \delta'')$ *and* $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$ *(as in Figure 1),* $\mathbf{P}' \rightsquigarrow \mathbf{P}$.

Refinement amounts in this case to the situation illustrated in example 5.3, namely, that while $\mathbf{P}'$ may allow extension of some carriers (corresponding to $\star'$ in the example), $\mathbf{P}$ may forbid it by introducing additional global guards. Thus, in general, all semantic functors for $\mathbf{P}$ are also semantic functors for $\mathbf{P}'$, but there may be some functors for $\mathbf{P}'$ which are not valid semantic functors for $\mathbf{P}$.

5.2.2 Vertical composition as an actualization of a particular semantic functor

There is, however, a more specific relation between the stepwise instantiation and the direct one. According to proposition 4.14, any semantic functor F_X for $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ induces a semantic functor F_Y for any instantiation of formal parameter $\mathbf{X}_\star$ by an actual parameter $\mathbf{Y}_\star$. If we now consider the results of respective actualizations, i.e., functors F_Z (obtained by stepwise actualization through $\mathbf{Y}_\star$ first along ν and then ρ) and F'_Z (obtained by direct actualization along $(\nu;\rho)$) which are both induced starting from the same, given F_X, then it turns out that the semantics is fully compositional, i.e., both functors are equal.

We discuss it in more detail. The semantic counterpart of the diagram from Figure 1 is shown below.



Given a semantic functor F_X (in the uppermost diagram), proposition 4.14 allows us to construct a functor F_Y, and similarly, an F_Z can be constructed given an arbitrary F'_Y. Thus, using F_Y

obtained from the actualization along ν for F'_Y, we can construct an F_Z from a given F_X. Notice that the associated ι_Z guarantees the image of $\mathsf{Mod}(\mathbf{Z}_\star)$ to be included in $\mathsf{Mod}(\mathbf{Z}_{\rho(-)})$.

For the direct actualization, we can obtain F'_Z from a given F_X by proposition 4.14. On the other hand, by fact 5.4, we also have the inclusion (functor) $\mathsf{i} : \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$. Hence, composing we obtain $\mathsf{F}_Z; \mathsf{i} : \mathsf{Mod}(\mathbf{Z}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$, which gives a possible semantic functor F'_Z for the PDT $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \delta''')$. Compositionality of actualization is expressed in the following proposition.

Proposition 5.6 *With the notation from the diagram above, where all functors are induced by* F_X *(in particular,* $\mathsf{F}_Y = \mathsf{F}'_Y$ *and* $\iota_Y = \iota_{Y1}$*):* $\mathsf{F}_Z; \mathsf{i} = \mathsf{F}'_Z$*.*

Proof. The discussion above shows that $\mathsf{F}_Z; \mathsf{i}$ is a possible semantic functor for $\mathbf{P}'$. To show the equality to F'_Z induced by a direct actualization, there remains a couple of tedious details.

I. Firstly, ι_{Z1}, associated with the functor F'_Z obtained from the direct actualization, will include $\mathsf{Mod}(\mathbf{Z}_\star)$ in $\mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$, while ι_Z associated with F_Z obtained through the stepwise actualization guarantees only inclusion in $\mathsf{Mod}(\mathbf{Z}_{\rho(-)})$. We show that it actually is a special case of a functor obtained from a direct actualization, i.e., that actually $\mathsf{F}_Z; |_{\mu''} : \mathsf{Mod}(\mathbf{Z}_\star) \to \mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$ – the lowest square (with two inclusions and reducts) commutes.

I.1. This is unproblematic when $\mathbf{Z}_{\rho(-)} = \mathbf{Z}_{(\nu;\rho)(-)}$, so let us consider the case when they are not equal. Then there is a global guard $x \prec c_s$ (of sort s) which is included in $\mathbf{Z}_{(\nu;\rho)(-)}$ but not in $\mathbf{Z}_{\rho(-)}$. This means that for any algebra $A \in \mathsf{Mod}(\mathbf{Z}_\star)$, $(\mathsf{F}'_Z(A))|_{\mu'''} \models x \prec c$ – in other words, F'_Z does not extend the carrier of s.

I.2. In principle, from the diagram, it might look that F_Z might extend this carrier since we may have $x \prec c_s \notin \mathbf{Z}_{\rho(-)}$. However, this last fact holds only if s is in the image of ρ (which makes the respective global guards disappear from $\mathbf{Z}_{\rho(-)}$). At the same time, since the guard is present in $\mathbf{Z}_{(\nu;\rho)(-)}$, it means that s is not in the image of $(\nu;\rho)$ – hence its ρ pre-image s' must not be in the image of ν.

I.3. This means that the respective guard $x \prec c_{s'} \in \mathbf{Y}_{\nu(-)}$ and, by the pushout construction, $x \prec c_{s'} \in \mathbf{P}[\mathbf{Y}]_\star$. But then the respective guard $x \prec \rho'(c_{s'}) = c_s$ will also appear in $\mathbf{P}[\mathbf{Z}]_\star$. Finally, since s' is not in the image of ν, we get $\mu'(c_{s'}) = c_{s'}$, which implies that also $\mu''(c_s) = c_s$. In short F_Z will not, after all, extend the carrier of sort s, and hence $(\mathsf{F}_Z(A))|_{\mu''} \in \mathsf{Mod}(\mathbf{Z}_{(\nu;\rho)(-)})$.

II. To prove the main claim, we need to look at the details of definitions of induced functors. What we have to show is that the following two are equal for any $A \in \mathsf{Mod}(\mathbf{Z}_\star)$ (cf. definition 4.13):

1. $\mathsf{F}'_Z(A) = \iota_{Z1}(A) \oplus_{\iota_{Z1}(A)|_{(\nu;\rho)}} \mathsf{F}_X(A|_{(\nu;\rho)})$ – direct actualization, and
2. $\mathsf{F}_Z(A) = \iota_Z(A) \oplus_{\iota_Z(A)|_\rho} \mathsf{F}_Y(A|_\rho)$, where $\mathsf{F}_Y(A|_\rho) = \iota_{Y1}(A|_\rho) \oplus_{\iota_{Y1}(A|_\rho)|_\nu} \mathsf{F}_X((A|_\rho)|_\nu)$.

The problem here might possibly originate from the situation as in fact 5.4.2 which was illustrated in example 5.3, i.e., that $\mathsf{F}'_Z(A)$ yields an algebra which does not satisfy the global guard $x \prec c$ satisfied by all algebras in $\mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star)$. Showing equality of 1. and 2. we show, in particular, that such a situation does not occur.

Actually it will suffice to show that $\iota_{Z1} = \iota_Z$, because the induced functor is constructed from F_X and ι (cf. def. 4.12, 4.13). This will, in particular, imply that $(\iota_{Z1}(A))|_{(\nu;\rho)} = ((\iota_Z(A))|_\rho)|_\nu$, for all algebras $A \in \mathsf{Mod}(\mathbf{Z}_\star)$ – the fact which is sufficient for concluding the equality of two functors according to def. 4.13. The table below lists the definitions of (the relevant) ι's induced by ι_X :

	$a.\quad \iota_{Z1}(A)$	$b.\quad \iota_Z(A)$	$c.\quad \iota_Y(A\vert_\rho)$
$\mathbf{S}:1.$	$if\ s \in \mathbf{S}_X:$ $(\nu;\rho)(s)^{\iota_{Z1}(A)} = s^{\iota_X(A\vert_{(\nu;\rho)})}$	$if\ s \in \mathbf{S}_Y:$ $\rho(s)^{\iota_Z(A)} = s^{\iota_Y(A\vert_\rho)}$	$if\ s \in \mathbf{S}_X:$ $\nu(s)^{\iota_Y(A\vert_\rho)} = s^{\iota_x((A\vert_\rho)\vert_\nu)}$
$2.$	$otherwise: s^{\iota_{Z1}(A)} = s^A$	$otherwise: s^{\iota_Z(A)} = s^A$	$otherwise: s^{\iota_Y(A\vert_\rho)} = s^{A\vert_\rho}$
$\Omega:3.$	$if\ \omega \in \Omega_X:$ $(\nu;\rho)(\omega)^{\iota_{Z1}(A)} = \omega^{\iota_X(A\vert_{(\nu;\rho)})}$	$if\ \omega \in \Omega_Y:$ $\rho(\omega)^{\iota_Z(A)} = \omega^{\iota_Y(A\vert_\rho)}$	$if\ \omega \in \Omega_X:$ $\nu(\omega)^{\iota_Y(A\vert_\rho)} = \omega^{\iota_X((A\vert_\rho)\vert_\nu)}$
$4.$	$if\ \omega \notin (\nu;\rho)[\Omega_X]: \omega(\overline{x})^{\iota_{Z1}(A)}$ $= \omega(\overline{x})^A,\ if\ all\ \overline{x} \in \vert A\vert$ $= \emptyset,\ otherwise$	$if\ \omega \notin \rho[\Omega_Y]: \omega(\overline{x})^{\iota_Z(A)}$ $= \omega(\overline{x})^A,\ if\ all\ \overline{x} \in \vert A\vert$ $= \emptyset,\ otherwise$	$if\ \omega \notin \nu[\Omega_X]: \omega(\overline{x})^{\iota_Y(A\vert_\rho)}$ $= \omega(\overline{x})^{A\vert_\rho},\ if\ all\ \overline{x} \in \vert A\vert_\rho\vert$ $= \emptyset,\ otherwise$

For any symbol $s \in \Sigma(\mathbf{Z}_\star)$ we have three possibilities:

1. it is not in the image of ρ (and hence not in the image of $(\nu;\rho)$ either), or
2. it is in the image of ρ but not of $(\nu;\rho)$, or
3. it is in the image of $(\nu;\rho)$.

To simplify the notation, we will ignore possible renamings, e.g., in case 2) we will assume that $s = \rho(s)$, and similarly, in case 3) that $s = \rho(s) = \rho(\nu(s))$. Justification of equations, written $\stackrel{Rc}{=}$, refers to row R, column c in the table above. We use functoriality of the reduct, i.e., the fact that $(A|_\rho)|_\nu = A|_{(\nu;\rho)}$, without mentioning it. Let us first consider the sorts.

1. $s^{\iota_{Z1}(A)} \stackrel{2a}{=} s^A \stackrel{2b}{=} s^{\iota_Z(A)}$
2. $s^{\iota_Z(A)} \stackrel{2b}{=} s^{\iota_Y(A|_\rho)} \stackrel{2c}{=} s^{A|_\rho} = s^A \stackrel{2a}{=} s^{\iota_{Z1}(A)}$
3. $s^{\iota_{Z1}(A)} \stackrel{1a}{=} s^{\iota_X(A|_{(\nu;\rho)})} = s^{\iota_X((A|_\rho)|_\nu)} \stackrel{1c}{=} s^{\iota_Y(A|_\rho)} \stackrel{1b}{=} s^{\iota_Z(A)}$

So, operations:

1. $\omega^{\iota_{Z1}(A)}(\overline{x}) \stackrel{4a}{=} \left\{ \begin{array}{ll} \omega^A(\overline{x}) & if\ \overline{x} \in |A| \\ \emptyset & otherwise \end{array} \right\} \stackrel{4b}{=} \omega^{\iota_Z(A)}(\overline{x}).$
2. By 4a. we have $\omega^{\iota_{Z1}(A)}$ as in the previous point. Since ω is not in the image of ν, we have $\omega^{\iota_Z(A)} \stackrel{3b}{=} \omega^{\iota_Y(A|_\rho)}(\overline{x}) \stackrel{4c}{=} \left\{ \begin{array}{ll} \omega^{A|_\rho}(\overline{x}) & if\ \overline{x} \in |A|_\rho| \\ \emptyset & otherwise \end{array} \right.$. But for all $\overline{x} : \overline{x} \in |A| \iff \overline{x} \in |A|_\rho|$, since ω, and hence all its sorts, are in the image of ρ. Then also $\omega^A(\overline{x}) = \omega^{A|_\rho}(\overline{x})$ which proves the equality $\omega^{\iota_Z(A)} = \omega^{\iota_{Z1}(A)}$ in this case.
3. $\omega^{\iota_{Z1}(A)} \stackrel{3a}{=} \omega^{\iota_X(A|_{(\nu;\rho)})} = \omega^{\iota_X((A|_\nu)|_\rho)} \stackrel{3c}{=} \omega^{\iota_Y(A|_\rho)} \stackrel{3b}{=} \omega^{\iota_Z(A)}$

$\square$

5.3 Horizontal composition

Definition 5.7 *For PDTs* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *and* $(\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$, *we define their horizontal composition to be* $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta;\delta'))$.

Proposition 5.8 *The composition as defined in 5.7 is (isomorphic to) a PDT. (In the sense that there exists a* $\mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star \simeq \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$ *such that* $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star, (\delta;\delta'))$ *is a PDT).*

Proof. The first 4 points of definition 3.15 are trivially satisfied. We have to verify point 5.

5a. If, for some $\star : \mu'(\mu(\star)) \neq \delta'(\delta(\star))$ then either 1) $\mu(\star) \neq \delta(\star)$ or 2) $\mu'(\star) \neq \delta'(\star)$.

Let us start with 1). By 5a of definition 3.15, $\mathbf{P}[\mathbf{Y}]_\star$ contains the axiom $\mu(\star) \prec \delta(\star)$. If both $c_1 = \mu(\star) \neq \star \neq \delta(\star) = c_2$, then this axiom is actually $c_1 \prec c_2$ and, by point 5b, $\mu'(c_1) \prec \mu'(c_2) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$. But then also both δ' and μ' are identities on c_1, c_2, i.e., this last axiom is in fact $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$.

If either $\delta(\star) = \star$ or $\mu(\star) = \star$, then we must have that $\delta(\star) = \star$ since, if $\mu(\star) = \star$, then by the presence of $\mu(\star) \prec \delta(\star)$ in $\mathbf{P}[\mathbf{X}]_\star$, we would have to have also $\delta(\star) = \star$. By point 5b, we have then $\mu'(\mu(\star)) \prec \mu'(\star) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$, while by 5a, $\mu'(\star) \prec \delta'(\star) = \delta'(\delta(\star)) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$. But then, adding the axiom $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$ yields an isomorphic specification.

Let us now consider the case 2). Having verified case 1), we can now assume that $\mu(\star) = \delta(\star)$. If $\mu(\star) = \delta(\star) = c \neq \star$, then μ', δ' are identities on c, which contradicts the assumption of this case. I.e., $\delta(\star) = \mu(\star) = \star$. But then by 5a, we have $\mu'(\star) \prec \delta'(\star) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$, which is the required axiom $\mu'(\mu(\star)) \prec \delta'(\delta(\star))$.

5b. This follows trivially: by 5b, $\mu(\phi) \in \mathbf{P}[\mathbf{X}]_\star$ and then, by the same point, $\mu'(\mu(\phi)) \in \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$.

5c. Follows equally easily. Let $x_1 \prec \star_1, \ldots, x_m \prec \star_m, \overline{a} \Rightarrow \overline{b}$ be a fully guarded axiom from $\mathbf{X}_\star$. Then, by 5c, $x_1 \prec \delta(\star_1), \ldots, x_m \prec \delta(\star_m), \overline{a} \Rightarrow \overline{b}$ is in $\mathbf{P}[\mathbf{X}]_\star$. But then, by the same point, $x_1 \prec \delta'(\delta(\star_1)), \ldots, x_m \prec \delta'(\delta(\star_m)), \overline{a} \Rightarrow \overline{b}$ is in $\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$. □

5.4 Horizontal composition – semantics

As was the case with vertical composition, horizontal composition of PDTs gives us a more structured specification. According to proposition 5.8, composing horizontally two PDTs, we obtain a new PDT with the associated class of semantic functors. We show that semantics of a PDT obtained by a stepwise, horizontal composition of PDTs $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and $\mathbf{P}' = (\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$, which can be written as $\mathbf{P};\mathbf{P}'$, is a refinement of the semantics of the respective composed PDT $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta;\delta'))$ – the former, possessing more structure in the form of the intermediary stage $\mathbf{P}[\mathbf{X}]_\star$, may put additional restrictions on the admissible functors. Yet, composition of such functors will always yield a functor for the composed PDT. We show this fact first.

Proposition 5.9 *Given PDTs $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and $(\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$, with the semantic functors $\mathsf{F}_\mathbf{X} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ and $\mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \to \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star)$. The composition $\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathbf{X}_\star \to \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$, is a semantic functor for $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta;\delta'))$.*

The proposition means that all the loops in the following diagram commute:

$$\begin{tikzcd}
\mathsf{Mod}(\mathbf{X}_\star) \arrow[d, "\iota_\mathbf{X}"] \arrow[dr, "\mathsf{F}_\mathbf{X}"] \arrow[drr, bend left, "\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}"'] \arrow[dd, bend right=40, "\iota'"'] & & \\
\mathsf{Mod}(\mathbf{X}_-) \arrow[d, dotted, "\iota_{\mathbf{P}[\mathbf{X}]}|_\mu"] & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[l, "|_\mu"'] \arrow[d, "\iota_{\mathbf{P}[\mathbf{X}]}"] \arrow[dr, "\mathsf{F}_{\mathbf{P}[\mathbf{X}]}"] & \\
\mathsf{Mod}(\mathbf{X}_-) & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_-) \arrow[l, dotted, "|_\mu"'] & \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \arrow[l, "|_{\mu'}"'] \arrow[ll, bend left, "|_{\mu;\mu'}"]
\end{tikzcd}$$

Proof. First notice that, by fact 3.18, μ is also a specification morphism $\mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_-$, and so $|_\mu$ is also a functor $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]_-) \to \mathsf{Mod}(\mathbf{X}_-)$. Thus $|_{\mu;\mu'} = |_{\mu'};|_\mu$ is a functor $\mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ as indicated on the diagram.

The functor $\iota' : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ corresponding to $\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}$ is defined by $\iota' = \iota_\mathbf{X};(\iota_{\mathbf{P}[\mathbf{X}]}|_\mu)$, i.e., $\iota'(A)$ is given by:

- $s^{\iota'(A)} = s^{\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}$, for sorts $s \in \Sigma(\mathbf{X}_\star)$
- $\omega^{\iota'(A)}(\overline{x}) = \omega^{\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}(\overline{x})$, for operations $\omega \in \Sigma_-(\mathbf{X}_\star)$
- $\star_s^{\iota'(A)} = (\mu;\mu')(\star_s)^{\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A)}$.

Thus, the outermost triangle commutes, i.e., for any $A \in \mathsf{Mod}(\mathbf{X}_\star) : (\mathsf{F}_\mathbf{X};\mathsf{F}_{\mathbf{P}[\mathbf{X}]}(A))|_{\mu;\mu'} = \iota'(A)$:

- sorts: $s^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P[X]}}(A))|_{\mu;\mu'}} = s^{\iota'(A)}$ since $\mu;\mu'(s) = s$
- operations: $\omega^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P[X]}}(A))|_{\mu;\mu'}} = \omega^{\iota'(A)}$ since $\mu;\mu'(\omega) = \omega$
- subsorts: $\star_s^{(\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P[X]}}(A))|_{\mu;\mu'}} = (\mu;\mu')(\star_s)^{\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P[X]}}(A)} = \star_s^{\iota'(A)}$

The tight monomorphism $\iota'_A : A \to \iota'(A)$ is defined $\forall A \in \mathsf{Mod}(\mathbf{X}_\star)$ by: $\iota'_A = \iota_A; (\iota_{PA}|_\mu)$, where ι_A and ι_{PA} are tight monomorphisms associated with $\iota_{\mathbf{X}}$ (and A) and $\iota_{\mathbf{P[X]}}$ (and $\mathsf{F}_{\mathbf{X}}(A)$), respectively. Since ι_{PA} is a tight monomorphism, then so is its reduct $\iota_{PA}|_{\mu'}$. Then, ι'_A, being a composition of two tight monomorphisms, is a tight monomorphism [15].

The conditions of fact 3.25 are satisfied, so we conclude that $\mathsf{F}_{\mathbf{X}};\mathsf{F}_{\mathbf{P[X]}}$ is indeed a semantic functor for the composed PDT. □

The following example illustrates that horizontal composition, introducing an intermediary parameter, can actually be a strict refinement of the composed PDT, i.e., that some functors admissible as a semantics for the composed PDT may no longer be obtained as a composition of the semantic functors for the component PDTs.

Example 5.10 *The following PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P[X]}_\star, \delta)$ requires extension of the parameter algebra A with a new function f and allows extending A's carrier with new elements (one of which may be d).*

$$\boxed{\begin{array}{ll} \multicolumn{2}{l}{\mathbf{X}_\star =} \\ \mathbf{S}: & El \\ S^\star: & \star : \to El \\ \Gamma: & x \prec \star \end{array}} \xrightarrow[\delta(\star)=\star]{\mu(\star)=ok} \boxed{\begin{array}{llrcl} \multicolumn{5}{l}{\mathbf{P[X]}_\star =} \\ \mathbf{S}': & El & & & \\ \Omega': & d: & & \to & El \\ & f: & El & \to & El \\ S^{\star\prime}: & \star, ok: & & \to & El \\ \Phi': & 1. & & & f(d) \doteq d \\ & 2. & x \prec \star & \Rightarrow & f(x) \doteq f(x) \\ & 3. & & & ok \prec \star \\ \Gamma': & 4. & & & x \prec \star \end{array}}$$

Let the semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P[X]}_\star)$ send an $A \in \mathsf{Mod}(\mathbf{X}_\star)$ to $\mathsf{F}(A)$ given by:

- $|\mathsf{F}(A)| = |A| \uplus d$*, i.e. d is a new element added to the carrier of A,*
- $f^{\mathsf{F}(A)}(x) = d$*, for all $x \in |\mathsf{F}(A)|$,*
- $ok^{\mathsf{F}(A)} = |A|$*, by the semantic functor requirement,*
- $\star^{\mathsf{F}(A)} = |\mathsf{F}(A)|$*, by default.*

Let's now introduce $\mathbf{W[X]}_\star$ as an intermediary parameter, i.e., we now have two PDTs $\mathbf{P}' = (\mu', \mathbf{X}_\star, \mathbf{W[X]}_\star, \delta')$ and $\mathbf{P}'' = (\mu'', \mathbf{W[X]}_\star, \mathbf{P[W[X]]}_\star, \delta'')$:

$$\mathbf{X}_\star \xrightarrow[\delta'(\star)=\star]{\mu'(\star)=\star} \boxed{\begin{array}{llrcl} \multicolumn{5}{l}{\mathbf{W[X]}_\star =} \\ \mathbf{S}'': & El & & & \\ \Omega'': & f: & El & \to & El \\ S^{\star\prime\prime}: & \star: & & \to & El \\ \Phi'': & 2. & x \prec \star & \Rightarrow & f(x) \doteq f(x) \\ \Gamma'': & 4. & & & x \prec \star_{EL} \end{array}} \xrightarrow[\delta''(\star)=\star]{\mu''(\star)=ok} \mathbf{P[X]}_\star = \mathbf{P[W[X]]}_\star$$

Obviously, we have that $\mathbf{P} = ((\mu';\mu''), \mathbf{X}_\star, \mathbf{P[W[X]]}_\star, (\delta';\delta''))$. But the refinement $\mathbf{P} \rightsquigarrow \mathbf{P}';\mathbf{P}''$ is strict – e.g., the functor F for the former cannot be obtained by composing any two functors for the latter two.

For any semantic functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{W[X]}_\star)$ can't extend the carrier of any $A \in \mathsf{Mod}(\mathbf{X}_\star)$, but merely adds a deterministic function f. Furthermore, any semantic functor $\mathsf{F}'' : \mathsf{Mod}(\mathbf{W[X]}_\star) \to \mathsf{Mod}(\mathbf{P[W[X]]}_\star)$ may add a new element d to the carrier of a parameter algebra $B \in \mathsf{Mod}(\mathbf{W[X]}_\star)$ and force $f(d) \doteq d$. However, F'' has to "preserve" the parameter algebra

B, i.e., B must be a tight subalgebra of $\mathsf{F}''(\mathsf{B})$. This means that function $f^{\mathsf{F}''(B)}$ applied to the elements from the carrier of B (i.e., from $ok^{\mathsf{F}''(B)}$) has to return elements from the same carrier (as it did in B). If d is a new element (as was the case for F), it will never be "reachable by f" from these old elements. This illustrates the impossibility of obtaining the original functor F as a composition of any F' and F''.

The following gives a more detailed and concrete example of the same idea of refining the structure of PDT by introducing an intermediary parameter.

Example 5.11 *We start with a specification of stacks from example 3.17 and refine it by introducing the intermediary parameter specification of constructed stacks. To do that we first refine the specification $\mathbf{Stack[El]}_\star \rightsquigarrow \mathbf{Stack[El]}'_\star$, by adding the the subsorts ok_{stack}, and error constants for stacks and elements:*

$$
\boxed{\begin{array}{ll}
\multicolumn{2}{l}{\textbf{spec El}_\star =}\\
\mathbf{S}: & El\\
S^\star: & \star_{El} :\to El\\
\Gamma: & x \prec \star_{EL}
\end{array}}
\xrightarrow[\delta(\star_{El}) = ok]{\mu(\star_{El}) = ok}
\boxed{\begin{array}{lllll}
\multicolumn{5}{l}{\textbf{spec Stack[El]}_\star =}\\
\mathbf{S}': & \multicolumn{4}{l}{Stack, El}\\
\Omega': & empty: & & \to Stack & \\
 & top: & Stack & \to El & \\
 & pop: & Stack & \to Stack & \\
 & push: & El \times Stack & \to Stack & \\
 & errStack & & \to Stack & \\
 & errEl & & \to Nat & \\
S^{\star\prime}: & \star_{El}, ok_{El}: & \to El & & \\
 & \star_{St}, ok_{St}: & \to Stack & & \\
\Phi': & 1. & & & empty \doteq empty\\
 & 2. & x \prec ok_{El}, s \prec \star_{St} & \Rightarrow & top(push(x,s)) \doteq x\\
 & 3. & x \prec ok_{El}, s \prec \star_{St} & \Rightarrow & pop(push(x,s)) \doteq s\\
 & 4. & & & pop(empty) \doteq errStack\\
 & 5. & & & top(empty) \doteq errEl\\
\Gamma': & 6. & & & x \prec \star_{EL}\\
 & 7. & & & s \prec \star_{St}
\end{array}}
$$

The intermediary parameter $\mathbf{ConstrStack[El]}_\star$ contains only the intended constructors for stacks::

$$
\mathbf{El}_\star \xrightarrow[\delta'(\star_{El}) = \star_{El}]{\mu'(\star_{El}) = \star_{El}}
\boxed{\begin{array}{llll}
\multicolumn{4}{l}{\textbf{spec ConstrStack[El]}_\star =}\\
\mathbf{S}': & \multicolumn{3}{l}{Stack, El}\\
\Omega': & empty: & & \to Stack\\
 & push: & El \times Stack & \to Stack\\
S^{\star\prime}: & \star_{El}: & \to El & \\
 & \star_{St}: & \to Stack & \\
\Phi': & 1. & empty \doteq empty & \\
\Gamma': & 6. & x \prec \star_{EL} & \\
 & 7. & s \prec \star_{St} &
\end{array}}
\xrightarrow[\substack{\mu''(\star_{St}) = ok_{St}\\ \delta''(\star_{St}) = ok_{St}}]{\substack{\mu''(\star_{El}) = ok_{El}\\ \delta''(\star_{El}) = ok_{El}}}
\mathbf{Stack[El]}_\star
$$

Let $\mathsf{F}' : \mathsf{Mod}(\mathbf{El}_\star) \to \mathsf{Mod}(\mathbf{ConstrStack[El]}_\star)$ and $\mathsf{F}'' : \mathsf{Mod}(\mathbf{ConstrStack[El]}_\star) \to \mathsf{Mod}(\mathbf{Stack[El]}_\star)$ be semantic functors for the respective PDTs. The effect of the intermediary parameter is that, in any algebra $C = \mathsf{F}''(\mathsf{F}'(A))$, both $empty^C$ and all stacks obtained by $push^C$ must belong to ok^C_{St}.

This reflects the more structured design which need not apply to a semantic functor of the original PDT $(\mu, \mathbf{El}_\star, \mathbf{Stack[El]}_\star, \delta)$, where empty could be any element of the sort Stack, possibly errStack which could (and should) be outside ok_{St}.

The specification $\mathbf{Stack[El]}_\star$ does not say anything about errStack or errEl but these can be new elements added to the carrier of $B = \mathsf{F}'(A)$. (This effect could be forced by the specification by adding the axioms $errStack \prec ok_{St} \Rightarrow$, and $errEl \prec ok_{El} \Rightarrow$.)

6 Refinement

We now summarize the concept of refinement of PDT. As we have emphasised, it amounts not only to the simple model class inclusion but, primarily, to introduction of additional *structure* on the PDTs. The following definition captures the general concept

Definition 6.1 *A PDT* $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$ *refines a PDT* $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, $\mathbf{P} \rightsquigarrow \mathbf{P}'$, *if there exist functors* $\mathsf{R}_\mathbf{X} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}'_\star)$ *and* $\mathsf{R}_{\mathbf{P}[\mathbf{X}]} : \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$, *such that for any semantic functor* F' *for* $\mathbf{P}'$, *the functor* $\mathsf{R}_\mathbf{X}; \mathsf{F}'; \mathsf{R}_{\mathbf{P}[\mathbf{X}]}$ *is a semantic functor for* $\mathbf{P}$.

The following diagram illustrates the requirement:

$$\begin{tikzcd}
 & \mathsf{Mod}(\mathbf{X}_\star) \arrow[dl, "\mathsf{R}_\mathbf{X}"'] \arrow[d, "\iota"'] \arrow[drr, dotted, "\mathsf{R}_\mathbf{X};\mathsf{F}';\mathsf{R}_{\mathbf{P}[\mathbf{X}]}"] & & \\
\mathsf{Mod}(\mathbf{X}'_\star) \arrow[d, "\iota'"'] \arrow[drr, "\mathsf{F}'"] & \mathsf{Mod}(\mathbf{X}_-) \arrow[r, phantom, "1."] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \arrow[ll, bend left=0, "|_\mu"'] \\
\mathsf{Mod}(\mathbf{X}'_-) \arrow[r, phantom, "2."] & & \mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \arrow[ll, "|_{\mu'}"] \arrow[ur, "\mathsf{R}_{\mathbf{P}[\mathbf{X}]}"'] &
\end{tikzcd}$$

The relation is trivially transitive, i.e., $\mathbf{P} \rightsquigarrow \mathbf{P}' \rightsquigarrow \mathbf{P}'' \Rightarrow \mathbf{P} \rightsquigarrow \mathbf{P}''$.

The contravariance of $\mathsf{R}_\mathbf{X}$ on the parameter side may seem a bit unusual since, following the idea of refinement as the model class inclusion, one might expect the refinement relation to hold also when $\mathbf{X}_\star \rightsquigarrow \mathbf{X}'_\star$, i.e., $\mathsf{Mod}(\mathbf{X}_\star) \supseteq \mathsf{Mod}(\mathbf{X}'_\star)$. However, one should keep in mind that we are talking about design specifications of actual structure of data types/programs. A PDT with source $\mathbf{X}'_\star$ could not, in general, replace a PDT with the source $\mathbf{X}_\star \rightsquigarrow \mathbf{X}'_\star$.

A simple case is when $\mathbf{P}[\mathbf{X}]_\star \rightsquigarrow \mathbf{P}[\mathbf{X}]'_\star$, i.e., when $\mathsf{R}_\mathbf{X}$ is identity and $\mathsf{R}_{\mathbf{P}[\mathbf{X}]}$ is a model class inclusion $\mathsf{Mod}(\mathbf{P}[\mathbf{X}]'_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ (as was the case of vertical composition in subsection 5.2.1).

Other examples were 5.10 and 5.11 in section 5.4, where both $\mathsf{R}_\mathbf{X}$ and $\mathsf{R}_{\mathbf{P}[\mathbf{X}]}$ were identities but where intermediary parameter forced additional requirements which did not (necessarily) follow from the original, refined PDT.

Finally, we give an example showing an even more particular case of refinement by adding structure, where the formal parameter of a PDT is itself refined to a PDT.

Example 6.2 *Let* $\mathbf{P} = (\mu, \mathbf{Set}_\star, \sqcup[\mathbf{Set}]_\star, \delta)$ *be the following PDT which extends a specification of* *sets with a nondeterministic choice.*

$$
\begin{array}{|l|}
\hline
\textbf{spec Set}_\star = \\
\begin{array}{rl}
\mathbf{S}: & Set, El \\
\Omega: & \emptyset : \quad \to Set \\
 & \dashv : \quad El \times Set \to Set \\
S^\star: & \star_{El} :\to El \\
 & \star_{Set} :\to Set \\
\Phi: & 1.\quad x \dashv (y \dashv S) \doteq y \dashv (x \dashv S) \\
 & 2.\quad x \dashv (x \dashv S) \doteq x \dashv S \\
\Gamma: & 3.\quad x \prec \star_{EL} \\
 & 4.\quad S \prec \star_{Set}
\end{array} \\
\hline
\end{array}
$$

$\mu(\star_{El}) = \delta(\star_{El}) = ok_{El}$

$\mu(\star_{Set}) = \delta(\star_{Set}) = \star_{Set}$

$$
\begin{array}{|l|}
\hline
\sqcup[\mathbf{Set}]_\star = \\
\begin{array}{rlcl}
\mathbf{S}: & Set, El & & \\
\Omega: & \emptyset : \quad \to Set & & \\
 & \dashv : \quad El \times Set \to Set & & \\
 & \sqcup : \quad Set \to El & & \\
S^\star: & ok, \star_{El} :\to El & & \\
 & \star_{Set} :\to Set & & \\
\Phi: & 1.\quad x \prec ok, y \prec ok & \Rightarrow & x \dashv (y \dashv S) \doteq y \dashv (x \dashv S) \\
 & 2.\quad x \prec ok & \Rightarrow & x \dashv (x \dashv S) \doteq x \dashv S \\
 & 3.\quad x \prec ok & \Rightarrow & \sqcup(x \dashv S) \prec ok \\
 & 4.\quad x \prec ok, z \prec \sqcup(x \dashv S) & \Rightarrow & z \doteq x, z \prec \sqcup(S) \\
\Gamma: & 5. & & x \prec \star_{EL} \\
 & 6. & & S \prec \star_{Set}
\end{array} \\
\hline
\end{array}
$$

We admit here **extending carrier** El, $\mu(\star_{El}) = ok$, *which is motivated by the possible need of a new, "error" element to be returned by* $\sqcup(\emptyset)$.

A possible semantic functor F *may send a* $\mathbf{Set}_\star$ *algebra* A *on the algebra* $\mathsf{F}(A)$ *where, for any nonempty set* S, *the operation* $\sqcup^{\mathsf{F}(A)}(S)$ *returns all the elements of the set* S. $\sqcup^{\mathsf{F}(A)}(\emptyset)$ *may return a new, "error" element* $\perp$, *added to the carrier of* A. *Adding this element to a set,* $\perp \dashv^{\mathsf{F}(A)} S$ *may then result in the empty set* $\emptyset^{\mathsf{F}(A)}$.

Obviously, the specification $\mathbf{Set}_\star$ *can be naturally parameterized by elements, i.e., we "extract" from it a parameter (sub)specification. We obtain then* $\mathbf{P}' = (\mu', \mathbf{El}_\star, \mathbf{Set}[\mathbf{El}]_\star, \delta')$, *where* $\mathbf{El}_\star$ *contains merely the sort* El *and the global guard* $x \prec \star_{El}$, *while* $\mathbf{Set}[\mathbf{El}]_\star$ *is exactly the same as* $\mathbf{Set}_\star$. μ' *and* δ' *are identities on* $\star_{El}$.

The point is now that the composed PDT $\mathbf{P}';\mathbf{P}$ *is a refinement of* $\mathbf{P}$ *according to definition 6.1.*

$$
\begin{tikzcd}
\mathsf{Mod}(\mathbf{El}_\star) \arrow[d, dotted] \arrow[dr, "\mathsf{F}'"] & & \mathsf{Mod}(\mathbf{Set}_\star) \arrow[ll, "|_{\mu'} \,=\, \mathsf{R}_{\mathbf{Set}}"'] \arrow[dl, dotted, dash, "id"'] \arrow[d, dotted] \arrow[dr, dotted, "\mathsf{G}"] & \\
\mathsf{Mod}(\mathbf{El}_-) & \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star) \arrow[l, "|_{\mu'}"'] \arrow[d, dotted] \arrow[dr, "\mathsf{F}"] & \mathsf{Mod}(\mathbf{Set}_-) & \mathsf{Mod}(\sqcup[\mathbf{Set}]_\star) \arrow[l, "|_\mu"'] \arrow[dl, dotted, dash, "id \,=\, \mathsf{R}_{\sqcup[\mathbf{Set}]}"] \\
 & \mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_-) & \mathsf{Mod}(\sqcup[\mathbf{Set}[\mathbf{El}]]_\star) \arrow[l, "|_\mu"'] &
\end{tikzcd}
$$

Since $\mu'(\star_{El}) = \star_{El}$ *the* $|_{\mu'}$ *reduct will, actually, return an* $\mathbf{El}_\star$*-algebra (and not only an* $\mathbf{El}_-$*-algebra).*

Now, given F' *and* F, *the functor* G *can be trivially chosen to be* F, *if we let* $\mathsf{R}_{\mathbf{Set}}$ *be identity. More generally, if* F' *were such that* $|_{\mu'};\mathsf{F}' = \mathsf{id}_{\mathbf{Set}_\star}$, *we will have that* $\mathsf{G} = \mathsf{R}_{\mathbf{Set}};\mathsf{F}';\mathsf{F};\mathsf{R}_{\sqcup[\mathbf{Set}]} = \mathsf{F}$.

However, even in such cases, the refinement consists in requiring a more structured data type, which consists of building first an algebra of sets over a given algebra of elements, and then an algebra with choice (i.e., composing two functors $\mathsf{F}';\mathsf{F}$*). In this sense, it is reasonable to call* $\mathbf{P}';\mathbf{P}$ *a refinement of* $\mathbf{P}$*.*

Moreover, the functor F' *will not, in general, be surjective on the objects, i.e., it may choose only a subclass of all* $\mathbf{Set}[\mathbf{El}]_\star$ *algebras. In this case, the application of the composition to all models of* $\mathbf{El}_\star$*,* $\mathsf{F}';\mathsf{F}(\mathsf{Mod}(\mathbf{El}_\star))$ *may result in fewer* $\sqcup[\mathbf{Set}[\mathbf{El}]]_\star$ *algebras than* $\mathsf{F}(\mathsf{Mod}(\mathbf{Set}[\mathbf{El}]_\star))$*, which is another reason for viewing this composition as a refinement of the original PDT.*

There is yet another possibility of viewing the above as an example of refinement. Suppose that we have implemented the PDT $(\mu, \mathbf{Set}_\star, \sqcup[\mathbf{Set}]_\star, \delta)$, i.e., we have a functor G. Then, having implemented also a functor F', we can compose it with G. Since F' will not, typically, be surjective on the objects, this composition will, in general, yield a smaller subclass of $\mathsf{Mod}(\sqcup[\mathbf{Set}]_\star)$ than the image of G.

In any case, we can view the above process as a gradual development of a design for the flat specification $\sqcup[\mathbf{Set}]_\star$. In the first step, we extract from it the parameter $\mathbf{Set}_\star$ which prescribes a more specific, structured implementation. In the second step, we again extract the parameter $\mathbf{El}_\star$, requiring even more structure. Viewed in this way, our setting gives a concrete specialization of the general concept of "constructor implementations" from [12]. A more detailed study of the methodological possibilities offered by our PDTs is left for a future work.

7 Conclusions

We have presented a framework for specifying parameterized data types. The syntax of PDTs is defined by a series of restrictions on the syntax of parameterized specifications, with the additional means for indicating the possibility of extending the carrier of the parameter algebras as well as the axioms of the parameter specification (to apply also to the "new" elements).

Semantics of PDTs is defined by a class of functors which satisfy a generalization of the classical persistency requirement – the parameter has to be a (tight) subalgebra of its image under the semantic functor. This generalization leads to a great flexibility in specifying PDTs which was illustrated by a series of examples.

We have re-stated and proved the theorems of vertical and horizontal composition of PDTs in our setting. An important concept emerging from these theorems concerns refinement of PDTs. We have given a general definition of such a refinement which is reflected in concrete examples primarily as introduction of additional structure into the specified PDT.

We view PDTs as design specifications which put requirements not only on the abstract (input-output) properties of the implementations but also on the actual structure of the implementation. In this way, our PDTs give a concrete realization of a more general concept of 'constructor specifications' from [12] which has recently been included into CASL [6] as 'architectural specifications'. Indeed, the suggested refinement of PDTs can be naturally seen as an example of program development based on constructor specifications where successive stages amount to splitting the original, loose specification into smaller pieces, according to the desired structure of the intended implementation.

Although we have presented the whole setting using the institution of multialgebras, it should be easy to recognize the generic aspect of our definitions and constructions. Entirely analogous extensions can be made on the top of any semi-exact institution where signatures contain means of expressing predicates (or subsorts). In this way, we have quite a general way of applying our setting which vastly extends the specific context of order sorted algebras used in [10]. We would also maintain that our syntactic requirements are simpler than those introduced in the above paper.

A point which certainly requires a further study concerns reasoning about PDTs. We expect that addition of generic axiom schemata expressing closure of the parameter algebras (i.e., that operations applied to the “old” elements return “old” elements) will lead to a complete axiomatization but this issue needs to be investigated.

On the other hand, we would like to use PDTs for study and, perhaps, design of more specific structuring mechanisms at the level of implementations. We believe that the current work can provide a useful starting point for designing more detailed constructs, for instance, for architectural specifications in languages like CASL.

References

[1] H. Ehrig, H.-J. Kreowski, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Parameter passing in algebraic specification languages. In *Proceedings of Workshop on Program Specification*, volume 134 of *LNCS*, 1982.

[2] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1*. Springer, 1985.

[3] Harald Ganzinger. Parametric specifications, parameter passing and optimizing implementations. Technical Report TUM-18110, Technical University of Munich, 1981.

[4] Joseph A. Goguen and R. M. Burstal. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.

[5] Joseph A. Goguen and Rod M. Burstall. Cat: a system for the structural elaboration of correct programs from structured specifications. Technical Report CSL-118, SRI International, 1980.

[6] CoFI. The Common Framework Initiative. *CASL – The Common Algebraic Specification Language*. http://www.brics.dk/Projects/CoFI/Documents/CASL.

[7] Yngve Lamo and Michał Walicki. Modeling partiality by nondeterminism - from abstract specifications to flexible error handling. Technical Report 178, Depeartment of Informatics, University of Bergen, 1999.

[8] Yngve Lamo and Michał Walicki. The instution of multialgebras. Technical Report 209, Department of Informatics, University of Bergen, 2000.

[9] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[10] Axel Poigné. Another look at parameterization using algebras with subsorts. In *Proceedings of MFPC*, volume 176 of *LNCS*. Springer, 1984.

[11] Donald Sanella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: parameterisation revisited. *Acta Informatica*, 29:689–736, 1992.

[12] Donald Sanella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica*, 25:233–281, 1988.

[13] Andrzej Tarlecki. Institutions: An abstract framework for formal specifications. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 4. Springer, 1999.

[14] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: parameterization and the power of specification techniques. In *Proceedings of POPL*. ACM, 1979.

[15] Michał Walicki and Marcin Białasik. Categories of relational structures. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*. Springer, 1998.

[16] Michał Walicki and Sigurd Meldal. Multialgebras, power algebras and complete calculi of identities and inclusions. volume 906 of *Lecture Notes in Computer Science*. Springer, 1995.

[17] Michał Walicki and Sigurd Meldal. Algebraic approaches to nondeterminism-an overview. *ACM Computing Surveys*, 29, 1997.