Electronic Notes in Theoretical Computer Science 70 No. 3 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume70.html 18 pages

Composition and Refinement of Specifications of Parameterized Data Types

Yngve Lamo [1]

Faculty of Engineering, Bergen University College, 5020 Bergen, Norway

Michał Walicki [2]

Department of Informatics, University of Bergen, 5020 Bergen, Norway

Abstract

In [5] we introduced a framework for specification of parameterized data types utilizing a generalization of the traditional semantics based on the pushout construction. In the present paper, we address the issue of program development using this framework with particular emphasis on the notion of refinement. Unlike for the loose specifications, refinement does not amount merely to a narrowing of the model class, but primarily to introduction of additional structure into the specified program. We give examples based on the analogues of the classical vertical and horizontal composition of such specifications.

1 Introduction

The need for modularization techniques in software development is well motivated by large software projects. At the implementation stage of such a large project, it is possible to identify subtasks of the whole system as parameter programs. By stepwise identification of subtasks, the whole system can be implemented by composition of parameterized programs.

The important distinction between parameterized specifications and specifications of parameterized programs has been originally pointed out in [8]. The major difference concerns the objects which are reused. *Parameterized specifications*, "PSPs", offer means to combine and reuse specification *texts*. This makes PSPs applicable for structuring the problem domain at the analysis stage of a software project. *A specification of a parameterized data type*,

[1] yla@hib.no

[2] michal@ii.uib.no. The second author gratefully acknowledges the financial support received from the Norwegian Research Council, NFR.

"PDT" [8,9], on the other hand, requires a reusable *implementation* of a program. Thus PDTs offer a formalism to reuse program pieces, i.e. to structure programs in a modular way. The model class of such a specification is seen as consisting of some – perhaps all – functors sending models of the formal parameter specification $\mathbf{X}$ to models of the parameterized specification $\mathbf{P}[\mathbf{X}]$:

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

Now, 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, namely, persistency of the functors from (1), e.g., [2,10,1]. This requirement is, however, very restrictive forbidding, in general, extension of the data types from the parameter program with new elements.

In [5] we have introduced a more adequate framework for specifying PDTs. We gave a generalization of the concept of persistent functors, so that our semantic functors can add new elements to the parameter algebras, but still ensuring its protection. This generalizes the idea from [6,7] (where only extension with new error elements was possible), by allowing extensions also with "regular" elements (e.g., extend a monoid to a group by adding inverse elements), and to choose whether the axioms from the parameter specification shall apply to the new elements or not.

The new contribution of this paper is the notion of refinement and results about composition of PDT's. We discuss refinement of PDTs in relation to the classical concept of implementation as model class inclusion. The main difference concerns our view of PDTs as design specifications, i.e., rather low level specifications which prescribe not only some desired functionality but also a specific *structure* to the program. Refinement amounts then to introduction of additional structure and exemplifies the idea of "constructor specifications" from [9]. We also present results about vertical and horizontal composition of PDTs which, in fact, provide examples of refinement. Their relation to the respective classical concepts is also discussed.

To make the paper at least to some extent self-contained, we summarize the contents of [5] in section 2. For even more details and proofs the reader is referred to [4]. Section 3 opens with a general concept of PDT refinement and then exemplifies it by vertical and horizontal composition. The presentation is based on the institution of multialgebras [3], but it should be easy to discern its more general applications. Section 4 contains some comments on this, and other concluding remarks.

2 Background

We work in the institution of multialgebras $\mathcal{MA}$, [3], allowing specifications of nondeterministic data types. But the only aspect used here is that $\mathcal{MA}$ allows one to interpret constants as sets (i.e., unary predicates).

Definition 2.1 Given a standard algebraic signature $\Sigma = (\mathbf{S}, \Omega)$, a Σ-multialgebra A is given by:

- a carrier set s^A for each sort symbol $s \in \mathbf{S}$
- an operation $\omega^A : s_1^A \times \ldots \times s_n^A \to \mathcal{P}(s^A)$ for each operation symbol $\omega : s_1, \ldots, s_n \to s \ \in \Omega$
- composition of operations is defined by pointwise extension: for a set $\overline{X} \subseteq \overline{s}^A : \omega^A(\overline{X}) = \bigcup_{\overline{x} \in \overline{X}} \omega^A(\overline{x})$

Terms over Σ and variables from X, $T(\Sigma, X)$ are defined in the standard way, and then any assignment $\alpha : X \to |A|$ of *individuals* from the carrier induces a unique interpretation $\alpha(t)$ of all terms $t \in T(\Sigma, X)$.

Definition 2.2 A specification $\mathbf{SP}$ consists of a signature Σ and a set of Σ-formulae which are either atoms:

- $t \doteq t'$ for $t, t' \in T(\Sigma, X)$
- $t \prec t'$ for $t, t' \in T(\Sigma, X)$

or general sequents: $a_1 \ldots a_n \Rightarrow b_1 \ldots b_m$, where each a_i, b_j is an atom.

Definition 2.3 The models of a specification $\mathbf{SP}$, $\mathsf{Mod}(\mathbf{SP})$ is the class of all multialgebras satisfying the axioms of $\mathbf{SP}$ according to the following.

Given an assignment $\alpha : X \to |A|$, $A \models_\alpha \phi$ iff:

- $A \models_\alpha t \doteq t'$ iff $\alpha(t) = \alpha(t') = \{e\}$ for some $e \in |A|$
- $A \models_\alpha t \prec t'$ iff $\alpha(t) \subseteq \alpha(t')$
- $A \models_\alpha a_1 \ldots a_n \Rightarrow b_1 \ldots b_m$ iff $\exists a_i : A \not\models_\alpha a_i$, or $\exists b_j : A \models_\alpha b_j$.

A satisfies ϕ, $A \models \phi$, iff $A \models_\alpha \phi$ for all α.

These pieces yield the institution $\mathcal{MA}$, where morphisms in the model categories are weak homomorphisms, i.e., satisfying $h(\omega^A(x)) \subseteq \omega^B(h(x))$. $\mathcal{MA}$ is semi-exact (in fact, exact), that is, the model functor sends pushouts of specifications onto pullbacks of the respective model classes. This provides a basis for the semantics of parameterized data types.

For specifying PDTs, we first introduce a special presentation of signatures. Each signature has a set of distinguished constant symbols $S^\star = \bigstar \cup C^\star$. The set $\bigstar$ contains constant symbols $\star_s$ for various sort symbols s, which are intended to denote all the elements of the respective sort s; $C^\star$ may contain additional constants representing subsorts.

Definition 2.4 A signature with sort constants, $\Sigma_\star$, is $(\mathbf{S}, \Omega, \bigstar, C^\star)$, where

$\Sigma = (\mathbf{S}, \Omega)$ is an ordinary signature, and $\bigstar \cup C^\star = S^\star$ is a (possibly empty) set of constants, such that $S^\star \cap \Omega = \emptyset$ and $\bigstar \cap C^\star = \emptyset$.

Definition 2.5 Given $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar, C^\star)$ we let:

(i) $\Sigma_- = (\mathbf{S}, \Omega, C^\star)$ – the reduced signature

(ii) $\Sigma = (\mathbf{S}, \Omega)$ – the standard (part of the) signature

Conversely, for an ordinary signature $\Sigma = (\mathbf{S}, \Omega)$, we let $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar, \emptyset)$, where $\bigstar = \{\star_s : s \textit{ for each } s \in \mathbf{S}\}$.

Saying "signature" we mean a signature with sort constants. When the distinction between $\bigstar$ and $C^\star$ does not matter, we write simply $S^\star$. (Signature morphisms map sort constants to sort constants.)

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 the elements in the parameter algebras – guards are needed to mark these elements.

Definition 2.6 For a $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar, C^\star)$, we distinguish the following:

(i) a *guard* γ is an atom $x \prec p$, with x a variable and $p \in S^\star = \bigstar \cup C^\star$;

(ii) a *(fully) guarded formula* is of the form $\gamma^*, \overline{a} \Rightarrow \overline{b}$ where
- $\overline{a}, \overline{b}$ are sequences of Σ_- atoms and
- γ^* is a sequence of guards $\gamma_i : x_i \prec p_i$ for each (and only) variable x_i occuring in the atoms $\overline{a}, \overline{b}$

γ_i's in the conditions of such formulae are called *local guards*;

A ground formula is guarded since it has no variables. The only places where $\bigstar$ may occur are in guards i, and hence in the antecedents of guarded formulae ii. Guards $x \prec \star$, where $\star \in \bigstar$, are only special cases – in general, $x \prec p$, where p is a any subsort constant, $p \in S^\star$, is a guard.

Definition 2.7 A *guarded specification* is $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$ where:

- $\Sigma_\star = (S, \Omega, \bigstar, C^\star)$ has (at least) one constant $\star_s \in \bigstar$ for each sort s.
- Each $\phi_\star \in \Phi_\star$ is a (fully) guarded formula.
- $\Gamma_\Sigma = \{x_s \prec \star_s : \star_s \in \bigstar\}$ is the set of axioms called *global guards* (with appropriately sorted variables as indicated by the subscript).

Note that all the local guards of the form $x \prec \star \Rightarrow ...$ in a guarded specification are trivially satisfied due to the presence of the global guards. Yet, this apparently redundant syntactic form will be of importance for defining the constructions on specifications. All such constructions will assume that the involved specifications are guarded.

Definition 2.8 Given a guarded specification $\mathbf{SP}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, its *weakening* is a specification $\mathbf{SP}_- = (\Sigma_\star, \Phi_\star)$.

Conversely, for an ordinary specification $\mathbf{SP} = (\Sigma, \Phi)$, $\mathbf{SP}_\star$ denotes the guarded specification $(\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, where $\bigstar$ is as in def. 2.5, $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar)$ and $\Gamma_\Sigma = \{x \prec \star_s : \star_s \in \bigstar\}$.

Keep 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_-.

Models for a guarded specification $\mathbf{SP}_\star$ are just the multialgebras (over signature $(\mathbf{S}, \Omega \cup S^\star)$) satisfying $\Phi_\star \cup \Gamma_\Sigma$. Given an ordinary specification $\mathbf{SP}$, there is an obvious equivalence of model categories between the unguarded $\mathsf{Mod}(\mathbf{SP})$ and guarded $\mathsf{Mod}(\mathbf{SP}_\star)$. Also, for a guarded $\mathbf{SP}_\star$ there is the obvious inclusion functor sending each algebra to itself:

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

2.1 Specification of parameterized data types

Definition 2.9 A parameterized data type specification (a PDT) is a quadruple $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, where

(i) $\mathbf{X}_\star = (\Sigma_\star, \Phi_\star, \Gamma_\Sigma)$, $\mathbf{P}[\mathbf{X}]_\star = (\Sigma'_\star, \Phi'_\star, \Gamma'_{\Sigma'})$ are guarded specs,

(ii) $\Sigma_\star = (\mathbf{S}, \Omega, \bigstar, C^\star) \subseteq (\mathbf{S}', \Omega', \bigstar', C^{\star\prime}) = \Sigma'_\star$ are signatures,

(iii) $\mu : \bigstar \to \bigstar' \cup C^{\star\prime}$ is called the *parameterization morphism*

(iv) $\delta : \bigstar \to \bigstar' \cup C^{\star\prime}$ is called the *local guard mapping*

(v) the two mappings, μ and δ are such that:
  (a) for every axiom $\phi_\star \in \Phi_\star : \delta(\phi_\star) \in \Phi'_\star$.
  (b) if $\mu(\star_s) \neq \delta(\star_s) \neq \star_s$ then $\mu(\star_s) \prec \delta(\star_s) \ \in \Phi'_\star$

For convenience, we treat μ and δ as signature morphisms $\Sigma_\star \to \Sigma'_\star$ which are identities on all symbols except (possibly) some of $\bigstar$ – this is reflected in point v.a, which means that if the respective axioms do not involove $\bigstar$, they are simply included in $\mathbf{P}[\mathbf{X}]_\star$. We write δ at the end of the tuple because in many constructions it plays no role, and then it may be dropped from the notation. For all practical purposes we can think of the syntax as given by μ and δ with $\delta(\star_s) = \mu(\star_s)$ or else $\delta(\star_s) = \star_s$ (see below). This covers most natural situations and will be the case in all our examples.

(v).(a) stated in more detail says: for each guarded axiom $\phi_\star \in \Phi_\star$:

$\phi_\star = x_1 \prec \star_1, \ldots, x_m \prec \star_m, x_{m+1} \prec p_{m+1}, \ldots, x_z \prec p_z, \overline{a} \Rightarrow \overline{b}$,

with all the local guards explicitly listed and $\star_i \in \bigstar$ (and $\overline{a}, \overline{b}$ sequences of Σ_- atoms), the corresponding axiom $\delta(\phi_\star) \in \Phi'_\star$, where:

$\delta(\phi_\star) = x_1 \prec \delta(\star_1), \ldots, x_m \prec \delta(\star_m), x_{m+1} \prec p_{m+1}, \ldots, x_z \prec p_z, \overline{a} \Rightarrow \overline{b}$.

Axioms of the form v.b are needed to ensure that the guarded axioms from the parameter specification will still apply, at least, to the elements originating from the parameter algebras. (We do not include the axiom $\mu(\star) \prec \delta(\star)$ when $\delta(\star) = \star$ to conform to the format from Def. 2.7 (and 2.6), but then it will be satisfied due to the global guards in $\mathbf{P}[\mathbf{X}]_\star$).

The image under $\mu : \bigstar \to S^{\star\prime}$ can be twofold:

1) $\mu(\star_s) = \star_s$, corresponds to the classical case of persistency, i.e., to **non-extending the carrier** s.
2) $\mu(\star_s) \neq \star_s$ introduces a distinction between the elements of sort s originating from the parameter, $\mu(\star_s)$, and the possibly new ones $\star_s$ – this allows for **extending carrier** s.

The mapping δ allows more flexibility in PDTs. If the carrier of sort s is not extended, case 1) above, δ has no effect – according to v.b, it has to be $\delta(\star_s) = \mu(\star_s) = \star_s$.[3] But if the carrier of s is 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 v.b, $\mu(\star_s) \prec \delta(\star_s)$, ensures that the old axioms still hold at least for the old elements.

2a) applies typically in situations when the carrier of a data type is extended with special kind of elements (like "error" values), the typical example being stacks parameterized by elements, where *pop*(*empty*) requires a new "error" element. 2b) applies in situations when the added elements are "essentially" of the same kind (e.g., group parameterized by monoid may require adding new, but "standard", inverse elements).

Proposition 2.10 *If* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *is a PDT then:*

(i) $\mu : \mathbf{X}_\star \to \mathbf{P}[\mathbf{X}]_\star$ *may not be a specification morphism, but*

(ii) $\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_-$ *is a specification morphism, and hence also*

(iii) $\mu : \mathbf{X}_- \to \mathbf{P}[\mathbf{X}]_\star$ *is a specification morphism.*

2.2 Semantics of PDTs

The semantics of PDTs is defined using a special case of the general (weak) homomorphisms of multialgebras.

Definition 2.11 A *tight* homomorphism $h : A \to B$ is a function such that for all operations ω: $h(\omega^A(x_1, ..., x_n)) = \omega^B(h(x_1), ..., h(x_n))$.

For constants, this means that $h(c^A) = c^B$, which also underlies the following logical characterization of a tight subalgebra.

Proposition 2.12 *If* $h : A \to B$ *is a tight* $\Sigma_\star$*-monomorphism and* ϕ *is an arbitrary, guarded* $\Sigma_\star$ *formula, then* $A \models \phi \iff B \models \phi$.

[3] In general, $\delta(\star_s)$ may be equal to another subsort constant p. But then v.b forces $\mathbf{P}[\mathbf{X}]_\star \models \star_s \prec p$ which, together with the global guard $x \prec \star_s$ says that the two constants, $\star_s$ and p, denote the same set, i.e., the whole carrier of sort s.

It is necessary to restrict the proposition to guarded formulae because assignments to "new" elements in B which are not in the image of $h[A]$ may actually falsify some formulae which hold in A.

Now, given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ 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) just before section 2.1,
- the composition $\mathsf{F}; \_|_\mu : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$.

Definition 2.13 The semantics of the PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ 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.

Such a functor is called a *semantic functor* for the PDT $\mathbf{P}$. The following fact is an alternative formulation of Def. 2.13.

Proposition 2.14 *A functor* $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ *is a semantic functor of a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *iff:*

(i) *there exists a functor* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_-)$ *such that* $\forall A \in \mathsf{Mod}(\mathbf{X}_\star)$ *there is a tight* $\Sigma_\star$*-monomorphism* $\iota_A : A \to \iota(A)$

(ii) *For every* $A \in \mathsf{Mod}(\mathbf{X}_\star) : \iota(A) = (\mathsf{F}(A))|_\mu$*, i.e., the following diagram commutes:*[4]

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

(iii) *For any* $\Sigma_\star$*-homomorphism* $h : A \to B$*,* F *satisfies* $\mathsf{F}; |_\mu(h) = h$*.*

In the category of multialgebras with weak (and also tight) homomorphisms, ι_A, being a monomorphism, is injective. The tightness 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 $s^{\mathsf{F}(A)}$ 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 its carrier 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 persistency is obtained as the special case when $\mu(\star_s) = \star_s$, for all s.

2.3 Actual parameter passing

Given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and an actual parameter passing ν with target $\mathbf{Y}_\star$, we would like to use the classical pushout approach to define the semantics of instantiation. To ensure the existence of the pushout, we have to ensure that the involved morphisms μ and ν are specification morphisms. The former will be so by Prop. 2.10. As to the latter, we have to take into account

[4] Although the ι here is not the same as in definition 2.13, its role is essentially the same and there is no real danger in confusing the two.

a more general situation, namely, the possibility that the actual parameter $\mathbf{Y}_\star$ contains sorts which are not in the image of ν. For such sorts, we should preserve the global guards in the pushout object. This motivates the following generalization of Def. 2.5.

Definition 2.15

(i) Given signatures $\Sigma_\star = (\mathbf{S}, \Omega, S^\star)$ and $\Sigma'_\star = (\mathbf{S}', \Omega', S^{\star\prime})$, 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 $\mathbf{SP}'_{\nu(-)} = (\Sigma'_\star, \Phi'_\star, \Gamma_{\Sigma'} \setminus \{x \prec \nu(\star_s) : s \in S\})$.

(ii) Given guarded specifications $\mathbf{X}_\star, \mathbf{Y}_\star$, an *actual parameter passing*, *app*, is a specification morphism: $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ satisfying $\nu(\star_s) = \star_{\nu(s)}$, for all $s \in \Sigma(\mathbf{X}_-)$, where $\Sigma(\mathbf{X}_-)$ is the signature of the specification $\mathbf{X}_-$.

The intuition behind the extra requirement $\nu(\star_s) = \star_{\nu(s)}$ is simply that since $\star_s$ denotes all elements of the sort s in the formal parameter $\mathbf{X}_\star$, its image under ν should do the same in the actual parameter $\mathbf{Y}_\star$.

Lemma 2.16 *Given (guarded) specifications* $\mathbf{X}_\star$ *and* $\mathbf{Y}_\star$*, if* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ *is an app, then* $\nu : \mathbf{X}_\star \to \mathbf{Y}_\star$ *is a specification morphism.*

Definition 2.17 Given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and an app $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ (with $\mathbf{X}_\star, \mathbf{Y}_\star$ guarded), the result of instantiation is a pushout $\mathbf{P}[\mathbf{Y}]_\star$ (in the category of specifications $\mathbf{Th}$ of $\mathcal{MA}$) of ν and μ:

$$\begin{array}{ccc} \mathbf{X}_- & \xrightarrow[\delta]{\mu} & \mathbf{P}[\mathbf{X}]_\star \\ {\scriptstyle\nu}\downarrow & & \downarrow{\scriptstyle\nu'} \\ \mathbf{Y}_{\nu(-)} & \xrightarrow{\mu'} & \mathbf{P}[\mathbf{Y}]_\star \end{array}$$

Since pushout is defined only up to isomorphism, the result need not be a PDT. Without going into details (which can be found in [4]), we only state here that it is possible to make a canonical choice of the pushout object which entails the following two facts.

Lemma 2.18 *Given a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$*, an app* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*, pushout as in the Def. 2.17 and the canonical pushout object: then* $\nu' : \mathbf{P}[\mathbf{X}]_- \to \mathbf{P}[\mathbf{Y}]_{\nu(-)}$ *is an actual parameter passing.*

Choosing also the induced δ' to be identical with μ', we obtain:

Proposition 2.19 *For a PDT* $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ *with an app* $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$*, one can choose the canonical pushout object* $\mathbf{P}[\mathbf{Y}]_\star$ *according to Def. 2.17, so that* $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star, \mu')$ *is a PDT.*

2.4 Semantics of instantiation

This is in one respect expressed in Prop. 2.19 – instantiation gives a specification of a new PDT. There is, however, another aspect which we call "actualization". Given a semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ and an app $\nu :$

$\mathbf{X}_{-} \to \mathbf{Y}_{\nu(-)}$, we want to *induce* a functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{Y}]_\star)$. This functor should satisfy the conditions for the semantics of PDTs (Prop. 2.14).

Definition 2.20 Given a PDT $(\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, the semantic functor $\mathsf{F} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ with the corresponding ι, and an app $\nu_{-} : \mathbf{X}_{-} \to \mathbf{Y}_{\nu(-)}$. A functor $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ is *induced by* ι iff for all $Y \in \mathsf{Mod}(\mathbf{Y}_\star)$:

(i) there is a tight monomorphism $\iota'_Y : Y \to \iota'(Y)$, and

(ii) $\iota(Y|_\nu) = \iota'(Y)|_\nu$ (and $\iota(h|_\nu) = \iota'(h)|_\nu$ for homomorphisms in $\mathsf{Mod}(\mathbf{Y}_\star)$)

The second point uses overloaded notion of ν which is admissible by lemma 2.16. It means the commutativity of the leftmost square 2.

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

By semi-exactness of $\mathcal{MA}$, the square 3. is a pullback, since the corresponding specification was constructed as a pushout according to definition 2.17. Amalgamation is used to define the functor F' given an induced ι'. Then the following proposition completes the picture ensuring the existence of an induced semantic functor F'.

Proposition 2.21 *Given a* $\iota : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{X}_{-})$ *associated with a semantic functor for a PDT and an app* $\nu : \mathbf{X}_{-} \to \mathbf{Y}_{\nu(-)}$*, there exist a functor* $\iota' : \mathsf{Mod}(\mathbf{Y}_\star) \to \mathsf{Mod}(\mathbf{Y}_{\nu(-)})$ *induced by* ι*.*

3 Refinement and composition

PDTs are design specifications imposing a particular structure on the specified program. Their refinement amounts thus not to a mere restriction of the model class but to introduction of *additional structure*.

Definition 3.1 A PDT $\mathbf{P}' = (\mu', \mathbf{X}'_\star, \mathbf{P}[\mathbf{X}]'_\star, \delta')$ refines $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$, $\mathbf{P} \rightsquigarrow \mathbf{P}'$, 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 $\mathbf{P}$.

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

The contravariance of $\mathsf{R}_{\mathbf{X}}$ on the parameter side in the refinement definition gives that the refinement $\mathbf{P}'$ has less semantical functors than the refined PDT $\mathbf{P}$, i.e. refinement of PDT's corresponds to a functorial subspace of semantic functors. It means that our refinement notion fits nicely with the traditional view of refinement as a subclass relation. Note however the difference between refinement of PDT's and refinement of flat specifications. Suppose that $\mathbf{X}_\star \rightsquigarrow \mathbf{X}'_\star$, i.e., $\mathsf{Mod}(\mathbf{X}_\star) \supseteq \mathsf{Mod}(\mathbf{X}'_\star)$. If this refinement is strict, i.e., $\mathsf{Mod}(\mathbf{X}_\star) \supset \mathsf{Mod}(\mathbf{X}'_\star)$, a semantic functor F' with source $\mathsf{Mod}(\mathbf{X}'_\star)$ could not, in general, be used in places were one assumes a functor with source $\mathsf{Mod}(\mathbf{X}_\star)$.

We will now review the concepts of vertical and horizontal composition of PDTs which will actually provide examples of refinement. The refinement will consist in that splitting a construction along a ρ into several steps $\rho_1; \ldots; \rho_n = \rho$ will not yield the same result as the direct construction along ρ but its refinement in the above sense. We show the compositionality theorems and discuss their relation to the classical concepts. Subsections 3.1 and 3.2 discuss vertical, and 3.3 horizontal composition.

Recall that, given a parameter passing diagram (like 1. below in Fig. 1), $(\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star, \mu')$ can be chosen to be a PDT by Prop. 2.19, and hence, $\mu' : \mathbf{Y}_- \to \mathbf{P}[\mathbf{Y}]_\star$ is a specification morphism by Prop. 2.10.

3.1 Vertical composition

Given two app's $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ and $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$, (as in the diagrams 1. and 2. in Fig. 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 app.

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. In general, we only have that $\mathbf{Z}_{(\nu;\rho)(-)} \models \mathbf{Z}_{\rho(-)}$. In spite of the possible differences, the following proposition ensures at least that composition of app's is still an app.

Proposition 3.2 *If $\nu : \mathbf{X}_- \to \mathbf{Y}_{\nu(-)}$ and $\rho : \mathbf{Y}_- \to \mathbf{Z}_{\rho(-)}$ are app's, then so is $(\nu;\rho) : \mathbf{X}_- \to \mathbf{Z}_{(\nu;\rho)(-)}$ (Fig. 1).*

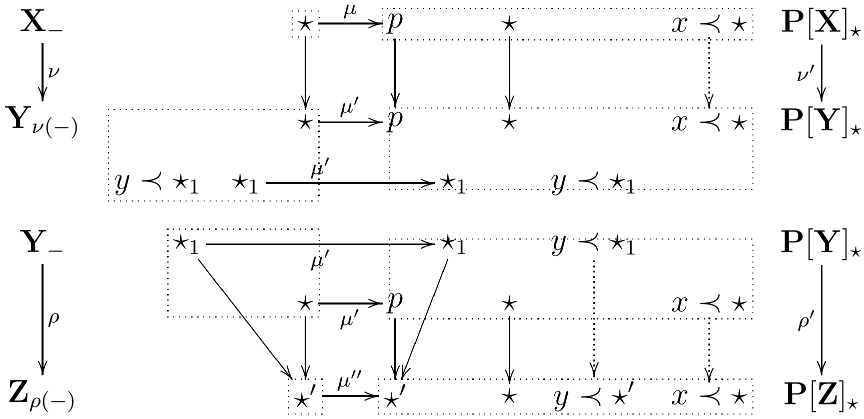
Now, in general, also the resulting $\mathbf{P}[\mathbf{Z}]_\star$ and $\mathbf{P}[\mathbf{Z}]'_\star$ may be different. In the classical case, this is because pushout is defined only up to isomorphism. We, however, may also drop and/or add some global guards on the

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

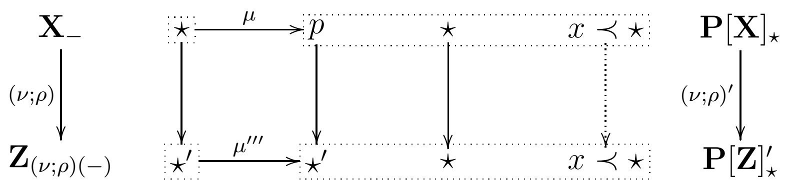
Fig. 1.

way. As in the case of $\mathbf{Z}_{\rho(-)}$ and $\mathbf{Z}_{(\nu;\rho)(-)}$, 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 3.3 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 $C^{\star'} = \{\star'\}$. So $(\mu'', \mathbf{Z}_{\star}, \mathbf{P}[\mathbf{Z}]_{\star}, \mathit{whoever})$ forbids extending the carrier $\star'$, while $(\mu''', \mathbf{Z}_{\star}, \mathbf{P}[\mathbf{Z}]'_{\star}, \mathit{whoever})$ does not.

Thus, $\mathbf{P}[\mathbf{Z}]_{\star}$ and $\mathbf{P}[\mathbf{Z}]'_{\star}$ need not be isomorphic. We only have:

Proposition 3.4 *With the notation from Fig. 1 and Ex. 3.3:*

(i) $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$.

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

This fact, that stepwise instantiation (along ν and then along ρ leading to $\mathbf{P}[\mathbf{Z}]_\star$) yields a different result than the direct instantiation (along $(\nu;\rho)$ 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. Hence we are interested in constructing new data types from others. Performing different constructions or, as in the case of vertical composition, performing constructions in different ways, may be expected to yield different results.

Stepwise instantiation, first along ν and then ρ, represents a slightly different construction than the direct instantiation along $\eta = \nu;\rho$. In fact, the former is a refinement of the latter according to Def. 3.1. 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 one step construction. This is indicated by Prop. 3.4.1 and we now illustrate the semantic aspect of this refinement.

3.2 Vertical composition – semantics

As noted before, the semantics of instantiation can be viewed 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 to the semantics of vertical composition.

3.2.1 Vertical composition as a refinement of PDTs.

A trivial, though by no means only, example of a PDT refinement from Def. 3.1 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 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 Prop. 3.4, $\mathbf{P}[\mathbf{Z}]_\star \models \mathbf{P}[\mathbf{Z}]'_\star$, i.e., we have an inclusion $\mathsf{i} : \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]_\star) \subseteq \mathsf{Mod}(\mathbf{P}[\mathbf{Z}]'_\star)$. 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

Proposition 3.5 *For* $\mathbf{P} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \mu'')$ *and* $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \mu''')$ *(as in Ex. 3.3),* $\mathbf{P}' \rightsquigarrow \mathbf{P}$.

Strictness of this refinement is illustrated by Ex. 3.3: 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.

3.2.2 The classical concept of vertical composition

The classical concept is different but, nevertheless, follows from the above. With reference to Fig. 1, one considers there $\mathbf{P_Y} = (\mu', \mathbf{Y}_\star, \mathbf{P}[\mathbf{Y}]_\star, \delta')$ to be an implementation of $\mathbf{P_X} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and also $\mathbf{P_Z} = (\mu'', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]_\star, \delta'')$ to be an implementation of $\mathbf{P_Y}$. The statement is that then $\mathbf{P_Z}$ is also an implementation of $\mathbf{P_X}$. The concept of implementation, however, does not coincide with our notion of refinement of PDTs because it allows restrictions of the source as well as target categories. This will be a special case of the semantic counterpart of the diagram from Fig. 1, when ν and ρ induce the respective reduct functors which are inclusions. We then get that any semantic functor for the resulting PDT $\mathbf{P_Z}$ has a source and target included in, respectively, the source and target of the semantic functors for $\mathbf{P_X}$.

3.2.3 Vertical composition as an actualization

There is, however, a more specific relation between the stepwise instantiation and the direct one. According to Prop. 2.21, 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 the 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. The semantic counterpart of Fig. 1 is shown in Fig. 2. ($\mathsf{M}(\mathbf{X})$ abbreviates $\mathsf{Mod}(\mathbf{X})$.)

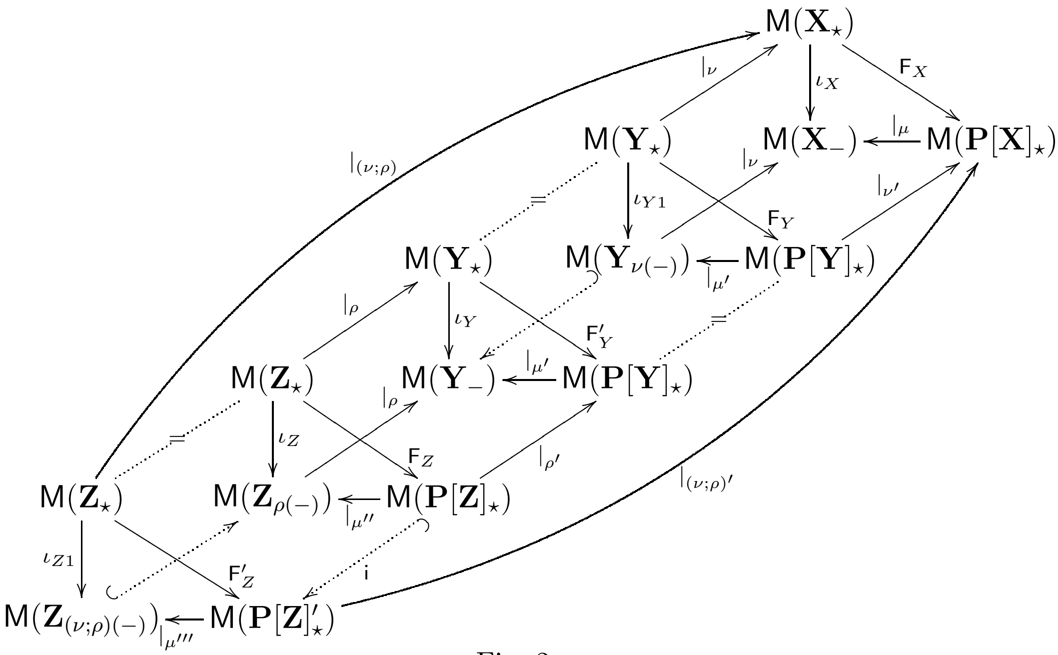


Fig. 2.

Given a semantic functor F_X (in the uppermost diagram of Fig. 2), Prop. 2.21 yields a functor F_Y, and similarly, an F_Z can be constructed given any 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 $\mathsf{Mod}(\mathbf{Z}_{\rho(-)})$.

Similarly, for the direct actualization, we can obtain F'_Z from a given F_X. By Prop. 3.4, 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 $\mathbf{P}' = (\mu''', \mathbf{Z}_\star, \mathbf{P}[\mathbf{Z}]'_\star, \mu''')$. Compositionality of actualization is expressed in the following:

Proposition 3.6 *With the notation from Fig. 2, with all functors induced by* F_X *(in particular,* $\mathsf{F}_Y = \mathsf{F}'_Y$ *and* $\iota_Y = \iota_{Y1}$*), we have:* $\mathsf{F}_Z;\mathsf{i} = \mathsf{F}'_Z$.

3.3 Horizontal composition

Definition 3.7 The horizontal composition of PDTs $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ and $\mathbf{P}' = (\mu', \mathbf{P}[\mathbf{X}]_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, \delta')$ is $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star, (\delta;\delta'))$.

We will denote it by $\mathbf{P};\mathbf{P}'$. Such a composition yields a new PDT.

Proposition 3.8 *The horizontal composition is (isomorphic to) a PDT –* $\exists(\mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star) \simeq \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$ *such that* $((\mu;\mu'), \mathbf{X}_\star, \mathbf{W}[\mathbf{P}[\mathbf{X}]]'_\star, (\delta;\delta'))$ *is a PDT.*

The first four points of Def. 2.9 are trivially satisfied and verification of point v is based, primarily, on the presence of axioms of the form v.b.

On the semantic side, the following proposition states that, given semantic functors for the (horizontally composed) component PDTs, their composition yields a semantic functor for the composed PDT.

Proposition 3.9 *Given* $\mathbf{P}, \mathbf{P}'$ *as in Def. 3.7 and semantic functors* $\mathsf{F}_{\mathbf{X}} : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star)$ *and* $\mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathsf{Mod}(\mathbf{P}[\mathbf{X}]_\star) \to \mathsf{Mod}(\mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star)$ *– then* $\mathsf{F}_{\mathbf{X}}; \mathsf{F}_{\mathbf{P}[\mathbf{X}]} : \mathbf{X}_\star \to \mathbf{W}[\mathbf{P}[\mathbf{X}]]_\star$ *is a semantic functor for* $\mathbf{P}; \mathbf{P}'$*.*

3.3.1 Horizontal composition as a refinement of PDTs.
Again, horizontal composition gives more structure. According to Prop. 3.8, composing horizontally two PDTs, we obtain a new PDT with the associated class of semantic functors. However, the semantics of a PDT obtained by a stepwise, horizontal composition of PDTs $\mathbf{P}$ and $\mathbf{P}'$ is a refinement of the semantics of the respective composed PDT $\mathbf{P}; \mathbf{P}'$. 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.

The following example illustrates this fact – that horizontal composition, introducing an intermediary parameter, can actually be a *strict* refinement of the composed PDT.

Example 3.10 The following PDT $\mathbf{P} = (\mu, \mathbf{X}_\star, \mathbf{P}[\mathbf{X}]_\star, \delta)$ 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).

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

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

- $|\mathsf{F}(A)| = |A| \uplus \{d\}$ – 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 introduce an intermediary parameter, i.e., we now have two PDTs $\mathbf{P}' = (\mu', \mathbf{X}_\star, \mathbf{W}[\mathbf{X}]_\star, \delta')$ and $\mathbf{P}'' = (\mu'', \mathbf{W}[\mathbf{X}]_\star, \mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star, \delta'')$, with $\mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star =$

$\mathbf{P}[\mathbf{X}]_\star$:

$$\mathbf{X}_\star \xrightarrow[\delta'(\star)=\star]{\mu'(\star)=\star} \boxed{\begin{array}{l} \mathbf{W}[\mathbf{X}]_\star = \\ \;\mathbf{S}'' : El \\ \;\Omega'' :\; f : El \to El \\ S^{\star\prime\prime} :\; \star : El \\ \;\Gamma'' : 3. \quad x \prec \star \end{array}} \xrightarrow[\delta''(\star)=\star]{\mu''(\star)=ok} \mathbf{P}[\mathbf{X}]_\star$$

Obviously, we have that $\mathbf{P} = \mathbf{P}';\mathbf{P}''$. But the stepwise composition from the last diagram is a strict refinement of $\mathbf{P}$ – the functor F cannot be obtained by composing any two functors for the latter two.

For any semantic functor $\mathsf{F}' : \mathsf{Mod}(\mathbf{X}_\star) \to \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star)$ can't extend the carrier of any $A \in \mathsf{Mod}(\mathbf{X}_\star)$, but merely adds a function f. Furthermore, any semantic functor $\mathsf{F}'' : \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star) \to \mathsf{Mod}(\mathbf{P}[\mathbf{W}[\mathbf{X}]]_\star)$ may add a new element d to the carrier of a parameter algebra $B \in \mathsf{Mod}(\mathbf{W}[\mathbf{X}]_\star)$ and force $f(d) = d$. However, F'' has to "preserve" the parameter algebra B, i.e., for $x \in ok^{\mathsf{F}''(B)}$, we must have $f^{\mathsf{F}''(B)}(x) \subseteq ok^{\mathsf{F}''(B)}$. If $d \notin ok^{\mathsf{F}''(B)}$ (as was the case for F), it will never be "reachable by f" from $ok^{\mathsf{F}''(B)}$. So we cannot obtain the original F as a composition of any F' and F''.

3.3.2 The classical concept of horizontal composition

This classical concept states that: if $\mathbf{X_1} \rightsquigarrow \mathbf{X_2}$ and $\mathbf{P_1}[\mathbf{X_1}] \rightsquigarrow \mathbf{P_2}[\mathbf{X_1}]$ then $\mathbf{P_1}[\mathbf{X_1}] \rightsquigarrow \mathbf{P_2}[\mathbf{X_2}]$, where all $\rightsquigarrow$ represent model class inclusions (in the opposite direction). This means that a functor for $\mathbf{P_2}[\mathbf{X_2}]$ may have a source $\mathsf{Mod}(\mathbf{X_2}) \subset \mathsf{Mod}(\mathbf{X_1})$, which makes it too specific to be used for obtaining a semantic functor for the original PDT with the parameter $\mathbf{X_1}$. Yet, this fact of "implementation commuting with parameterization" allows to perform independent refinements on various components ensuring that their composition will yield an implementation of the composition of the original components.

Although this property does not reflect our notion of refinement, it still obtains in our setting. If $\mathbf{P_1}[\mathbf{X_1}]_\star \rightsquigarrow \mathbf{P_2}[\mathbf{X_1}]_\star$ then, as remarked in section 3.2.1, we obtain the refinement of the respective PDTs. If, in addition, we have $\mathbf{X_1}_\star \rightsquigarrow \mathbf{X_2}_\star$, then any semantic functor for $(\mu_{21}, \mathbf{X_2}_\star, \mathbf{P_2}[\mathbf{X_2}]_\star, \delta_{21})$ is a restriction of the semantics of $(\mu_1, \mathbf{X_1}_\star, \mathbf{P_1}[\mathbf{X_1}]_\star, \delta_1)$, that is, an implementation in the classical sense in that the source and target categories of the semantic functors for the former are subcategories of, respectively, source and target categories of the semantic functors for the latter.

$$\begin{tikzcd}
 & \mathsf{Mod}(\mathbf{P_1}[\mathbf{X_1}]_\star) & \\
\mathsf{Mod}(\mathbf{X_1}_\star) \arrow[ur, "\mathsf{F}_1"] \arrow[rr, "\mathsf{F}_{21}"'] & & \mathsf{Mod}(\mathbf{P_2}[\mathbf{X_1}]_\star) \arrow[ul, hook, dotted, "\mathsf{i}"'] \\
 & \mathsf{Mod}(\mathbf{P_1}[\mathbf{X_2}]_\star) \arrow[uu, hook, "|_{\nu_1}"] & \\
\mathsf{Mod}(\mathbf{X_2}_\star) \arrow[uu, hook, "|_{\nu}"'] \arrow[ur, "\mathsf{F}'_1"] \arrow[rr, "\mathsf{F}'_{21}"] & & \mathsf{Mod}(\mathbf{P_2}[\mathbf{X_2}]_\star) \arrow[uu, hook, "|_{\nu_2}"']
\end{tikzcd}$$

The diagram illustrates this situation where, given an app $\nu : \mathbf{X_{1\star}} \to \mathbf{X_{2\star}}$ such that $|_\nu : \mathsf{Mod}(\mathbf{X_{2\star}}) \to \mathsf{Mod}(\mathbf{X_{1\star}})$ is an inclusion, the $|_{\nu_2}$, obtained by pullback, is an inclusion, too.

4 Conclusions

We have summarized the framework for specifying PDTs presented originally in [5]. The framework's syntax provides the means for indicating the possibility of extending the carrier of the parameter algebras as well as restricting the axioms of the parameter specification (to apply only to the "old" 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.

Viewing PDTs as design specifications, which put requirements not only on the abstract (input-output) properties of the implementation but also on its actual structure, we introduced a concept of refinement of PDTs which corresponds to introduction of more structure. In this way, our PDTs give a concrete realization of a more general concept of 'constructor specifications' from [9]. The 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. This refinement notion has been illustrated by the examples of vertical and horisontal composition.

The results have been presented using the institution of multialgebras. Yet, although not entirely institution independent, they can be easily repeated for many common institutions, as long as they satisfy a few requirements: they are semi-exact (admit amalgamation lemma); signatures contain symbols which in the model classes are interpreted as unary predicates; signature morphisms respect this distinction, i.e., send such predicate symbols only on such symbols; the model classes are concrete categories where monomorphisms are injective. Although the list may seem rather restrictive, most commonly used institutions do satisfy these requirements. The results, in particular Prop. 2.21, hold not only for $\mathcal{MA}$, but also for institutions of total algebras (with predicates), partial algebras (with predicates), membership algebras.

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 remains 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 also believe that the current work can provide a useful basis for designing more detailed constructs, for instance, for architectural specifications in languages like CASL.

References

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

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

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

[4] Yngve Lamo and Michał Walicki. Specification of parameterized data types. Technical Report 210, Department of Informatics, University of Bergen, 2000.

[5] Yngve Lamo and Michał Walicki. Specification of parameterized data types: Persistency revisited. *Nordic Journal of Computing*, 8:278–303, 2001.

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

[7] Axel Poigné. Error handling for parameterized data types. In *Proceedings of 3rd WADT*, volume 116 of *LNCS*. Springer, 1984.

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

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

[10] 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.