An approach to parameterized first-order specifications: semantics, correctness, parameter passing

Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Reif, Wolfgang. 1993. "An approach to parameterized first-order specifications: semantics, correctness, parameter passing." In Formal methods in programming and their applications: International Conference Academgorodok, Novosibirsk, Russia, June 28 – July 2, 1993; proceedings, edited by Dines Bjørner, Manfred Broy, and Igor V. Pottosin, 67–80. Berlin: Springer. https://doi.org/10.1007/bfb0039701.

An Approach to Parameterized First-order Specifications: Semantics, Correctness, Parameter Passing

Wolfgang Reif *
University of Karlsruhe
reif@ira.uka.de

Abstract

This paper presents an alternative approach to loose specifications of parameterized data types. The specification language is full first-order logic, and the semantics is a particular class of functions, mapping parameter algebras to (parameter generated) target algebras. We investigate monomorphicity and correctness of parameterized first-order specifications, and present simple syntactic criteria for these notions. Furthermore, correctness of standard parameter passing is studied. We give a characterization as well as a simple sufficient criterion for specifications with correct parameter passing. Finally, the interaction between parameter passing and correctness of specifications is investigated: We present conditions under which actualization of parameterized specifications preserves the correctness of the constituent specifications.

1 Introduction

Since the pioneering work of Liskov, Zilles, the ADJ-group and many others in the early seventies, a number of different approaches to algebraic specification have emerged and gained a strong influence on software engineering. In combination with logics of programs, algebraic specifications have become an indispensable tool for the formal description and construction of verified software. Usually, three main stream approaches are distinguished according to their semantics: initial algebra semantics ([LZ 74], [Gu 76], [GTW 78], [TWW82], [Ga 83], [Pa 85], [EM 85], [Pa 90], [EM 90]), terminal algebra semantics ([Wa 79], [HR 80], [Ka 83]) and loose semantics ([GGM 76], [BDPPW 79], [WB 82], [WPPDB 83], [CIP 87], [Wi 90]).

In the first two approaches a specification is meant to define a specific abstract data type precisely up to isomorphism, and therefore the semantics of a specification is a particular

---

* Author's address: Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, Postfach 6980, W-7500 Karlsruhe, FRG, Tel. +721-608-4245. This research was partly sponsored by the BMFT-project KORSO.

class of isomorphic algebras (initial or final algebras resp.). In the case of initial semantics the specification language is in general restricted to universal equations or Horn clauses, in order to guarantee the existence of initial models (for the extension by constraints see [Pa 90], [EM 90]). In the loose approach, however, a specification is rather thought of as a collection of requirements, open to a number of different interpretations. The semantics of a specification is therefore a class of possibly non-isomorphic interpretations. This view of a specification is an appropriate basis for software specification and development, and is adopted in this paper. The loose approach has some practical advantages: The specification language is full first-order logic and is not restricted to Horn clauses. This increases the flexibility in practical applications considerably. Furthermore, first-order specifications exhibit better "deductive properties" than Horn clause specifications, in the sense that in general more consequences can be deduced from the explicit axioms of the specification (see [Re 92]). This is a good basis for verification systems.

In this paper we present a parameterization mechanism for loose first-order specifications. It combines a pushout-style parameterization on the syntactic level (e.g. [EKTWW 80], [BG 80], [Eh 82], [Ga 83], [EM 85]) with a semantics based on classes of functions (cf. [SST 92]). A parameterized specification PSPEC = ($SPEC_p$, SPEC) is a pair of two first-order specifications. $SPEC_p$ is the *parameter specification* and SPEC is the *target specification*, the former beeing a subspecification of the latter. A single parameterized data type satisfying PSPEC is modelled by a partial function $\mathcal{F}$, that takes a parameter algebra $\mathcal{M}$ and returns a ("parameter generated") target algebra $\mathcal{F}(\mathcal{M})$. Roughly, the semantics SEM(PSPEC) is the class of all functions satisfying PSPEC. This loose semantics differs from the ones given in ASL ([SW 83], [Wi 86]) or Extended ML (for functor specifications, [ST 88]), but is similar to the one given in [SST 92] for specifications of parametric algebras. There are also similarities to PLUSS [Gau 92].

In general, loose specifications admit non-isomorphic interpretations, and therefore are ambiguous. In the software development process this is exploited in abstract requirement specifications. Only the crucial properties of the system are specified, leaving out implementational or other details. In later phases of the development however, the specifications tend to restrict the classes of described models more and more, ending up with design specifications describing an interpretation up to isomorphism. Specifications with this property are called *monomorphic*. If a data type $\mathcal{F}$ satisfies a monomorphic specification PSPEC, then PSPEC is called *correct* for $\mathcal{F}$. In this paper the main stress is put on how to detect monomorphicity, and how monomorphic and correct specifications can be developed. We investigate both notions, and present a simple proof theoretic criterion for monomorphic specifications. Furthermore, we investigate the actualization of parameterized specifications. We define the notion of *strict persistency* which characterizes the correctness of *standard parameter passing* (actualization with an unparameterized specification). We present a simple sufficient criterion for strict persistency, and show that for strictly persistent parameterized specifications, actualization is a correctness preserving operation.

The paper is organized as follows: In the next section we recall some basic definitions, and define simple and parameterized specifications. In section 3 we investigate the semantics of parameterized specifications. Section 4 is devoted to monomorphic and correct specifications, and section 5 investigates standard parameter passing.

2 Basic Definitions

2.1 Simple and parameterized specifications

A *signature* SIG = (S, OP) consists of a set of sorts S and a set OP of function- and predicate symbols over S. S* are the finite words over S with empty word λ. For $w = s_1..s_n$ OP_w is the set of n-ary predicate symbols with the indicated typing, and $OP_{w,s}$ is the set of function symbols with range sorts w and target sort s. For every sort $s \in S$ there is a countably infinite reservoir X_s of individual variables. X is the system of all these variable sets. For SIG = (S, OP) and X, $T_{OP,s}(X)$ denotes the *terms* of sort s over OP and X. $T_{OP,s}$ is the corresponding set of ground terms (= $T_{OP,s}(\emptyset)$). $T_{OP}(X)$ and T_{OP} are the sets of all terms and ground terms, respectively. A signature SIG = (S, OP) is a *subsignature* of SIG' = (S', OP') if $S' \supseteq S$ and $OP' \supseteq OP$. L(SIG, X) and $\forall$L(SIG, X) denote the sets of first-order and universal fomulas over SIG and X, respectively . A *simple* (unparameterized) specification SPEC = (S, OP, T) consists of a signature SIG = (S, OP) and a set T of first-order formulas. SPEC is a *subspecification* of SPEC' = (S', OP', T') if (S, OP) is a subsignature of (S', OP') and $T \subseteq T'$.

Parameterized specifications are based on *parameterized signatures* SIG = ($S \cup S_p$, $OP \cup OP_p$). We distinguish between *parameter sorts* S_p, *target sorts* S, *parameter operations* OP_p, and *target operations* OP. The sets are assumed to be disjoint, and $SIG_p = (S_p, OP_p)$ is called the *parameter signature*. Sometimes we use the notation $SIG_p \cup (S , OP)$ for SIG. A parameterized specification PSPEC = ($SPEC_p$, SPEC) consists of two simple specifications: the *parameter specification* $SPEC_p$ and the *target specification* SPEC, the former being a subspecification of the latter.

IND(SIG) denotes the set of first-order instances of a schema for *structural induction* over the target sorts S of SIG = ($S \cup S_p$, $OP \cup OP_p$). IND(SIG) is defined as follows: Let $\phi = \langle \varphi_s : s \in S \rangle$ a vector of formulas φ_s and $\mathbf{x} = \langle x_s: s \in S \rangle$ a vector of induction variables x_s of sort $s \in S$. The variable x_s occurs free in φ_s, and $\varphi_s(c)$ stands for substituting c for x_s in φ_s. We define IND(SIG) = { $(\text{Basis}(\mathbf{x}, \phi) \wedge \text{Hyp}(\mathbf{x}, \phi)) \rightarrow \text{Concl}(\mathbf{x}, \phi) : \phi = \langle \varphi_s : s \in S \rangle, \mathbf{x} = \langle x_s: s \in S \rangle$}. We set $\text{Basis}(\mathbf{x}, \phi) := \bigwedge_{c \in OP_{\lambda,s}} \varphi_s(c)$, $\text{Concl}(\mathbf{x}, \phi) := \bigwedge_{s \in S} \forall x_s.\varphi_s$ and $\text{Hyp}(\mathbf{x}, \phi) := \bigwedge \{\text{Step}(f, \mathbf{x}, \phi): f \in OP_{w,s}, s \in S\}$. For $f \in OP_{w,s}$, $s \in S$, with $w = s_1..s_r$, and new variables $x_1, .., x_r$ Step(f, x, ϕ) is the following formula:

$$\forall x_1,.., x_r. ((\bigwedge \{\varphi_{s'}(x') : \langle s', x' \rangle \in \{\langle s_1, x_1 \rangle, .., \langle s_r, x_r \rangle\}, s' \in S\}) \rightarrow \varphi_s(f(x_1, .., x_r))).$$

2.2 Semantic concepts

For SIG = (S, OP) a *SIG-algebra* $\mathcal{A} = ((A_s)_{s \in S}, (f_{\mathcal{A}})_{f \in OP})$ consists of non-empty carrier sets A_s and the interpretations $f_{\mathcal{A}}$ for the symbols from OP. The interpretation $f_{\mathcal{A}}$ is a total function or a predicate depending on what kind of symbol f is. We use a relaxed notation for algebras by just enumerating the components, for example, ($\mathbb{N}$, 0, succ, <) for the natural numbers with zero, successor and the less-than predicate. The class of SIG-algebras is abbreviated by Alg(SIG). We write $\mathcal{A} \cong \mathcal{B}$, if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. If SIG = (S, OP) is a subsignature of SIG', and $\mathcal{A}$ a SIG'-algebra, then the *reduct* of $\mathcal{A}$ wrt. SIG,

$\mathcal{A}|_{SIG}$, is the algebra $\mathcal{B}$ with $B_s = A_s$ for all $s \in S$ and $f_{\mathcal{B}} = f_{\mathcal{A}}$ for all $f \in OP$. Let SIG a subsignature of SIG', $\mathcal{A}$ a SIG'-algebra, and g an isomorphism from $\mathcal{A}|_{SIG}$ to $\mathcal{M}$. Then a SIG'-algebra $(\mathcal{A} \nabla_{SIG,g} \mathcal{M})$ can be defined, where roughly $\mathcal{A}|_{SIG}$ is replaced in $\mathcal{A}$ by $\mathcal{M}$ according to g. We call this algebra the *isomorphic alteration* of $\mathcal{A}$ by $\mathcal{M}$ wrt. g. We get the properties $(\mathcal{A} \nabla_{SIG,g} \mathcal{M}) \cong \mathcal{A}$ and $(\mathcal{A} \nabla_{SIG,g} \mathcal{M})|_{SIG} = \mathcal{M}$. Since in the sequel the actual isomorphism g is not important, we use the relaxed notation $(\mathcal{A} \nabla_{SIG} \mathcal{M})$.

The evaluation of a term $t \in T_{OP}(X)$ in $\mathcal{A}$ under the *valuation* $s: X \to \mathcal{A}$ is denoted by $t_{s,\mathcal{A}}$. We write $\mathcal{A}, s \models \varphi$ if the formula φ is true in $\mathcal{A}$ under s, $\mathcal{A} \models \varphi$ if $\mathcal{A}$ is a model of φ, and $T \models \varphi$ if every model of the formula set T is a model of φ. Alg(T) is the class of all models of T. An algebra is generated if each carrier element can be denoted by a ground term. Gen(T) is the class of generated models of T. For SPEC = (S, OP, T) Alg(SPEC) := Alg(T) and Gen(SPEC) := Gen(T).

Signature morphisms, *forgetful operations*, *specification morphisms* and *free functors* for equational parameterized specifications are defined as usual (see [EM 85] for details). A signature morphism h: SIG → SIG', maps sorts and operations of SIG to sorts and operations of SIG', a specification morphism h : SPEC → SPEC' is a signature morphism, which additionally guarantees that SPEC' entails the *translated axioms* of SPEC: SPEC' $\models$ h(SPEC). For a given signature morphism h: SIG → SIG' the forgetful operation V_h turns a SIG'-Algebra $\mathcal{A}$ into a SIG-algebra $\mathcal{B} := V_h(\mathcal{A})$ with $B_s := A_{h(s)}$ and $f_{\mathcal{B}} := h(f)_{\mathcal{A}}$.

3 Semantics of Parameterized First-order Specifications

We illustrate the syntax of parameterized specifications by the following description of finite sets over an arbitrary partially ordered domain: Sets_over_Partial_order is described in a compact notation for parameterized specifications. It refers to the parameter specification Partial_order and lists the target sorts, functions, predicates, variables and axioms. The sort *data*, and the partial ordering< are the only parameter sorts and operations. The target sort is *set*, the target operations are the empty set ∅, the insertion of a data element (insert), the membership predicate (∈), and an ordering (<<) on sets, a variant of the multiset ordering.

parameterized specification Sets_over_Partial_order

parameter	Partial_order
sorts	set
functions:	$\emptyset : \to$ set
	insert : data × set → set
predicates	infix $\in$: data × set
	infix << : set × set
vars	n, m : set
axioms	$\neg\ x \in \emptyset$
	$x \in \text{insert}(y, m) \leftrightarrow (x = y \vee x \in m)$
	$n = m \leftrightarrow \forall x. (x \in n \leftrightarrow x \in m)$
	$n << m \leftrightarrow \forall x. (x \in n \rightarrow \exists y. (y \in m \wedge x < y))$

end

parameter specification Partial_order
sorts data
predicates infix $<$: data $\times$ data
vars x, y, z : data
axioms $\neg x < x$
$x < y \wedge y < z \rightarrow x < z$
end

For simple specifications we adopt the loose semantics approach of Giarratana et al. ([GGM 76]) and the Munich CIP-group ([BDPPW 79]). The semantics of a simple specification SPEC is the class of its generated models Gen(SPEC). Our semantics of a parameterized specification PSPEC = ($SPEC_p$, SPEC) with signature SIG = ($S \cup S_p$, $OP \cup OP_p$) is based on the class of *parameter generated* or *S_p-generated* algebras, PGen(SIG). A SIG-algebra $\mathcal{A}$ is S_p-generated if each element of any target carrier (A_s for $s \in S$) can be constructed from the parameter carriers (A_s for $s \in S_p$) by finitely many applications of the target functions from OP. S_p-generated algebras are models of IND(SIG). We define the *target models* of PSPEC, Tar(PSPEC), to be the class of S_p-generated models of SPEC.

The interpretations of PSPEC are functions from Alg($SPEC_p$) to Tar(PSPEC). For example, the function *Set* mapping each model $\mathcal{M}$ of Partial_order to the algebra of finite sets over $\mathcal{M}$ (with the corresponding operations) is an interpretation of the above sample specification. However, we are not interested in arbitrary functions. We require that they preserve their parameters -like *Set* does-, or at least preserve them up to isomorphism. In general we cannot expect that the functions are defined for each parameter algebra, like *Set*, since the target specification might be inconsistent. Therefore, we require that they are defined whenever possible. This means that the functions are defined for a parameter algebra $\mathcal{M}$ iff there exists a target model $\mathcal{A}$ with $\mathcal{A}|_{SIG_p} \cong \mathcal{M}$. Finally, we require that the functions are compatible with isomorphism: The images of isomorphic arguments are again isomorphic.

3.1 Definition (Semantics of parameterized specifications)

Let PSPEC = ($SPEC_p$, SPEC) with signature SIG = ($S \cup S_p$, $OP \cup OP_p$). Then SEM(PSPEC) is the class of all partial functions $\mathcal{F} \in Alg(SIG_p) \rightarrow Alg(SIG)$ (with domain $dom(\mathcal{F})$ and range $rg(\mathcal{F})$) satisfying the following properties:

1) $rg(\mathcal{F}) \subseteq Tar(PSPEC)$
2) $dom(\mathcal{F}) = Tar(PSPEC)|_{SIG_p}$
3) for all $\mathcal{M} \in dom(\mathcal{F})$: $\mathcal{F}(\mathcal{M})|_{SIG_p} \cong \mathcal{M}$
4) for all $\mathcal{M}, \mathcal{M}' \in dom(\mathcal{F})$: $\mathcal{M} \cong \mathcal{M}' \Rightarrow \mathcal{F}(\mathcal{M}) \cong \mathcal{F}(\mathcal{M}')$ ■

Properties 1) and 2) guarantee that the images of a function $\mathcal{F}$ are target models, and that $\mathcal{F}$ is defined whenever possible. We call a function $\mathcal{F}$ with property 3) *persistent*. Property

4) states that $\mathcal{F}$ is compatible with isomorphism. An algebra $\mathcal{A}$ *satisfies* a simple specification SPEC if $\mathcal{A} \in$ SEM(SPEC). A function $\mathcal{F}$ *satisfies* PSPEC if $\mathcal{F} \in$ SEM(PSPEC). We call two functions $\mathcal{F}_1$, $\mathcal{F}_2$ *isomorphic* ($\mathcal{F}_1 \cong \mathcal{F}_2$), if $\mathrm{dom}(\mathcal{F}_1) = \mathrm{dom}(\mathcal{F}_2)$ and $\mathcal{F}_1(\mathcal{M}) \cong \mathcal{F}_2(\mathcal{M})$ for all $\mathcal{M} \in \mathrm{dom}(\mathcal{F}_1)$. The functions of SEM(PSPEC) are defined for $\mathrm{Alg}(\mathrm{SPEC_p})$ if each $\mathcal{M} \in \mathrm{Alg}(\mathrm{SPEC_p})$ can be extended to a parameter generated model of SPEC, i.e. $\mathrm{Tar}(\mathrm{PSPEC})|_{\mathrm{SIG_p}} = \mathrm{Alg}(\mathrm{SPEC_p})$. A specification PSPEC with this property is called *freely extendible.*

3.2 Remark (Elementary properties of SEM(PSPEC) and related work)

SEM(PSPEC) is closed under isomorphism: If $\mathcal{F}_1 \in$ SEM(PSPEC) and $\mathcal{F}_1 \cong \mathcal{F}_2$ then $\mathcal{F}_2 \in$ SEM(PSPEC). However, since we allow full first-order specifications, it might happen that the target specification SPEC is inconsistent or has no parameter generated models ($\mathrm{Tar}(\mathrm{PSPEC}) = \emptyset$). Then (and only then) SEM(PSPEC) consists of the undefined function. If Tar(PSPEC) is not empty, then SEM(PSPEC) in general contains different non-isomorphic functions. In our example, however, SEM(Sets_over_Partial_order) is a single class of isomorphic functions. How this special case can be detected from the axioms is shown in section 4.

With the initial semantics approach to equational parameterized specifications (see f.e. [EM 85]), the above semantics shares the view of parameterized data types as mappings from parameter algebras to target algebras. In the initial semantics approach, the free functors are always defined for all parameter algebras, but may fail to be persistent. In our approach, however, the functions are persistent per definition but possibly not defined for every parameter algebra. The functions satisfying freely extendible specifications are persistent and defined for $\mathrm{Alg}(\mathrm{SPEC_p})$. In practice we are interested in freely extendible specifications.

In ASL ([SW 83], [Wi 86]) the semantics of a parameterized specification is a mapping that takes a class of (parameter) algebras and returns a class of (result) algebras. In Extended ML ([ST 88]) the semantics of a functor specification is a function mapping an algebra to a class of algebras. Both are different from the semantics given above. In [SST 92] Sanella, Sokolowski and Tarlecki introduce a parameterization mechanism for *parametric algebras*, using a semantics similar to one given above.

4 Monomorphicity and Correctness

In this section we investigate the notions of monomorphicity and correctness, give a syntactic criterion how to detect momonorphicity from the axioms, and illustrate it by an example. We concentrate on parameterized specifications. For simple specifications we refer to [Wi 90], [Re 92].

4.1 Definition (Monomorphic specifications)

A parameterized specification PSPEC is *monomorphic* if any two functions $\mathcal{F}_1$ and $\mathcal{F}_2 \in$ SEM(PSPEC) are isomorphic. ■

The following analysis of monomorphic specifications is based on the fact that a PSPEC is monomorphic iff any two target models $\mathcal{A}$ and $\mathcal{B}$ with identical parameter reduct ($\mathcal{A}|_{SIG_p} = \mathcal{B}|_{SIG_p}$) are isomorphic. The latter criterion can be equivalently reformulated in terms of quasi-ground literals.

4.2 Definition (Quasi-ground literals)

Let $SIG = (S \cup S_p, OP \cup OP_p)$ be a parameterized signature with parameter subsignature $SIG_p = (S_p, OP_p)$. Let X_p be a set of variables for S_p. *Quasi-ground literals* are formulas of the shape: $t_1 = t_2$, $\neg t_1 = t_2$, $q(t_1,..,t_n)$ and $\neg q(t_1,..,t_n)$, where $t_1, t_2,.., t_n \in T_{OP}(X_p)$ and q is a predicate symbol from OP. Non-negated literals are called *positive*. ■

The term *quasi-ground literals* stems from the fact that the terms occuring in the literals are ground with respect to the target sorts, but may contain variables of the parameter sorts. Henceforth, a term $t \in T_{OP}(X_p)$ is called a *quasi-ground* term.

4.3 Fact (Monomorphicity and quasi-ground literals)

PSPEC is monomorphic $\Leftrightarrow$

for all $\mathcal{A}, \mathcal{B} \in$ Tar(PSPEC) with $\mathcal{A}|_{SIG_p} = \mathcal{B}|_{SIG_p}$, all positive quasi-ground literals φ, and for all valuations $s : X_p \to \mathcal{A}|_{SIG_p}$, the following holds: $\mathcal{A}, s \models \varphi \Leftrightarrow \mathcal{B}, s \models \varphi$. ■

Roughly speaking, a specification is monomorphic if it determines the truth of the positive quasi-ground literals for target models with identical parameter reducts. Monomorphic specifications arising in practice, often meet an even stronger condition: They determine the truth of the positive quasi-ground literals for all models with identical parameter reducts, that satisfy the target specification and the structural induction principle. This situation is studied in [Re 91] and leads to the following criterion for monomorphic specifications.

4.4 Theorem (Sufficient criterion for monomorphicity)

Let PSPEC = ($SPEC_p$, SPEC) with signature $SIG = SIG_p \cup (S, OP)$. PSPEC is monomorphic, if for each positive quasi-ground literal $\varphi(x_1, .., x_n)$ there exists a formula $\varphi'(x_1, .., x_n) \in L(SIG_p, X_p)$ such that:

$SPEC \cup IND(SIG) \models \varphi(x_1, .., x_n) \leftrightarrow \varphi'(x_1, .., x_n)$. ■

Now we turn to the correctness of specifications. A simple specification SPEC is *correct* for an algebra $\mathcal{A}$, if $\mathcal{A}$ is the only generated model of SPEC up to isomorphism. Correspondingly, PSPEC is correct for a function $\mathcal{F}$, if PSPEC describes $\mathcal{F}$ uniquely up to isomorphism.

4.5 Definition (Correctness of specifications)

A parameterized specification PSPEC = ($SPEC_p$, SPEC) is *correct* for $\mathcal{F} \in Alg(SIG_p) \rightarrow Alg(SIG)$, if $\mathcal{F} \in SEM(PSPEC)$ and PSPEC is monomorphic. ■

Note that if Tar(PSPEC) = $\emptyset$, PSPEC is correct for the undefined function. In order to prove that PSPEC is correct for $\mathcal{F} \in Alg(SIG_p) \rightarrow Alg(SIG)$, the following criterion based on 4.4 and 4.5 can be used.

4.6 Corollary (Correctness of specifications)

A parameterized specification PSPEC = ($SPEC_p$, SPEC) is correct for $\mathcal{F} \in Alg(SIG_p) \rightarrow Alg(SIG)$, if the following conditions hold:

1) for all $\mathcal{M} \in dom(\mathcal{F})$: $\mathcal{F}(\mathcal{M})$ is S_p-generated
2) for all $\mathcal{M} \in dom(\mathcal{F})$: $\mathcal{F}(\mathcal{M}) \models SPEC$
3) $dom(\mathcal{F}) \supseteq Tar(PSPEC)|_{SIG_p}$
4) for all $\mathcal{M} \in dom(\mathcal{F})$: $\mathcal{F}(\mathcal{M})|_{SIG_p} \cong \mathcal{M}$
5) for all positive quasi-ground literals $\varphi(x_1, .., x_n)$ there exists $\varphi'(x_1, .., x_n) \in L(SIG_p, X_p)$ such that $SPEC \cup IND(SIG) \models \varphi(x_1, .., x_n) \leftrightarrow \varphi'(x_1, .., x_n)$ ■

4.7 Example (Sets_over_Partial_order revisited)

With the help of 4.4 we show that Sets_over_Partial_order is monomorphic. To this purpose we classify the positive quasi ground literals: set equations t = t', predicate expressions $x \in t$ and $t << t'$. The terms t and t' take the form insert(x_1, insert(x_2, ..insert(x_k, $\emptyset$)..)) and insert(y_1, insert(y_2, ..insert(y_u, $\emptyset$)..)), respectively, and x_i and y_i are variables of the sort data. If the quasi-ground literal φ is $x \in t$ then φ' is $\bigvee_{i=1..k} x = x_i$ (*false* if k = 0). If φ is the equation t = t' then φ' is the formula $\forall z.\ ((\bigvee_{i=1..k} z = x_i) \leftrightarrow (\bigvee_{i=1..u} z = y_i))$, and if φ is $t << t'$, we set φ' $\forall z.\ ((\bigvee_{i=1..k} z = x_i) \rightarrow \exists z'.\ ((\bigvee_{i=1..u} z' = y_i) \wedge z < z'))$. In all cases $SPEC \cup IND(SIG) \models \varphi \leftrightarrow \varphi'$ holds.

Furthermore, by inspection of the conditions 1) to 4) of 4.6, we see that Sets_over_Partial_order, is correct for the function *Set* with dom(*Set*) = Alg(Partial_order) and $Set(\mathcal{M}) := (Pot_\omega(M), M, \emptyset, insert, \in, <<, <)$ for all $\mathcal{M} \in dom(Set)$. $Pot_\omega(M)$ denotes the finite subsets of M (carrier of $\mathcal{M}$) and the operations have the standard meaning. ■

The criteria 4.4 and 4.6 are useful to prove monomorphicity or correctness of a given specification. Example 4.7 illustrates the general technique to prove that the equality and the predicates are fixed by the axioms with respect to the parameter. However, for large specifications, this proof is very extensive. The higher the number of symbols in the signature, the more quasi-ground literals exist. Furthermore, from a methodological point of view, it is unrealistic to develop large specifications as a monolithic block. What is needed are structuring operations enabling the incremental design of correct specifications from smaller ones, preserving the correctness of the constituent specifications. Two operations that are available in almost all specification languages (CLEAR [BG 80], ASL [SW 83], [Wi 86], ACT ONE [EM 85], Extended ML [ST 88], SPECTRUM [Br 91]) are the disjoint union of specifications, and the enrichment of a specification with new axioms for new symbols. A large correct specification can be developed by starting out with a small fragment, proving it correct using the above criteria, and then extending it incrementally to the intended specification. Due to space limitations, we refer to [Re 91] for the criteria for union and enrichment.

5 Parameter Passing

In this section we study two questions: First, the correctness of standard parameter passing for parameterized full first-order specifications, and, second, the interaction between parameter passing and the correctness of the involved specifications. With standard parameter passing we mean actualization with simple specifications ([EM 85]).

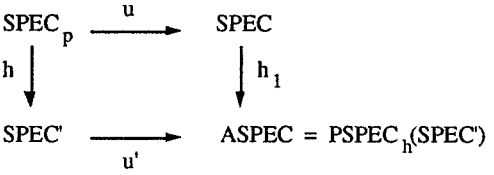
Roughly, the actualization of a parameterized data type $\mathcal{F}$ with an actual parameter $\mathcal{M}$ is the function application of $\mathcal{F}$ to $\mathcal{M}$. $\mathcal{M}$ needs not necessarily be formulated over the parameter signature of $\mathcal{F}$. It suffices if there is a signature morphism h between the parameter signature of $\mathcal{F}$ and the signature of $\mathcal{M}$. The result of the actualization of $\mathcal{F}$ by $\mathcal{M}$ and h is denoted by $\mathcal{F}_h(\mathcal{M})$. Syntactic actualization of PSPEC = ($SPEC_p$, SPEC) with an actual parameter specification SPEC' replaces $SPEC_p$ in SPEC by SPEC'. The connection between $SPEC_p$ and SPEC' is described by a specification morphism h. The resulting simple specification is denoted by $PSPEC_h$(SPEC').

Correctness of parameter passing is the following problem : Does $\mathcal{F}_h(\mathcal{M})$ satisfy $PSPEC_h$(SPEC') if $\mathcal{M}$ and $\mathcal{F}$ satisfy SPEC' and PSPEC respectively ? And conversely, can every $\mathcal{A}'$ satisfying $PSPEC_h$(SPEC') be obtained by actualization of a function $\mathcal{F}$ satisfying PSPEC with a parameter $\mathcal{M}$ satisfying SPEC'? If both answers are positive for all specification morphisms then parameter passing is correct for PSPEC. We provide a solution to this problem by a characterization of the specifications with correct parameter passing and give a syntactic criterion for this class.

The second problem solved in this section is the interaction between parameter passing and the correctness of the involved specifications: We present sufficient conditions under which for all specification morphisms h, $PSPEC_h$(SPEC') is correct for $\mathcal{F}_h(\mathcal{M})$, provided PSPEC and SPEC' are correct for $\mathcal{F}$ and $\mathcal{M}$, respectively .

5.1 Definition (Syntactic actualization)

Standard parameter passing requires three things: A parameterized specification PSPEC = ($SPEC_p$, SPEC) with signature SIG = ($S \cup S_p$, $OP \cup OP_p$), a simple specification SPEC' = (S', OP', T') with signature SIG', being the *actual parameter*, and a specification morphism h : $SPEC_p \rightarrow$ SPEC' which indicates which sorts and operations of $SPEC_p$ are replaced by which sorts and operations of SPEC'. Then the *value specification* $PSPEC_h(SPEC')$ with signature ($S' \cup S$, $OP' \cup OP^*$) results from replacing SPEC' for $SPEC_p$ in SPEC, thereby replacing the symbols of $SPEC_p$ that also may occur in SPEC according to h. OP* contains the same symbols like OP, but the typing of the symbols may have changed according to h. In the literature this syntactic parameter passing is illustrated with the following (pushout) parameter passing diagram (see [EM 85] for formal details):



The specification morphism h is called the *passing morphism*, u and u' are the *formal* and *actual parameter inclusions*. h_1 is the *induced passing morphism* which acts like h for symbols from $SPEC_p$ and is the identity for the target symbols of SPEC. Additionally, it is required that S and S' are disjoint as well as OP and OP'. ■

In the sequel we assume such a standard parameter passing diagram and refer to the above notation. The semantic actualization is based on *amalgamated sums* : If $\mathcal{M}' \in$ Alg(SPEC') and $\mathcal{A} \in$ Alg(SPEC) and $V_h(\mathcal{M}') = \mathcal{A}|_{SIG_p} =: \mathcal{M}$, then the amalgamated sum $\mathcal{B} := \mathcal{M}' +_{\mathcal{M}} \mathcal{A}$ is uniquely defined in Alg(ASPEC) with $\mathcal{B}|_{SIG'} = \mathcal{M}'$ and $V_{h1}(\mathcal{B}) = \mathcal{A}$.

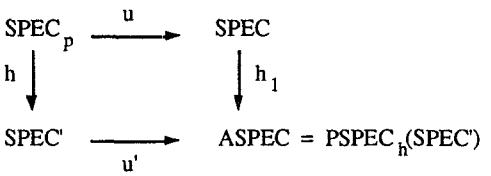
5.2 Definition (Semantic actualization)

Let PSPEC = ($SPEC_p$, SPEC), SPEC' and the specification morphism h be given like above. For $\mathcal{F} \in$ SEM(PSPEC) and $\mathcal{M}' \in$ Alg(SPEC') with $\mathcal{M} := V_h(\mathcal{M}') \in dom(\mathcal{F})$ we define the *actualization* of $\mathcal{F}$ by $\mathcal{M}'$ and h: $\mathcal{F}_h(\mathcal{M}') := \mathcal{M}' +_{\mathcal{M}} (\mathcal{F}(\mathcal{M}) \nabla_{SIG_p} \mathcal{M})$. ■

Since $\mathcal{F} \in$ SEM(PSPEC) is persistent per definition, $\mathcal{F}(\mathcal{M})|_{SIG_p}$ and $\mathcal{M}$ are isomorphic, and hence $\mathcal{F}(\mathcal{M}) \nabla_{SIG_p} \mathcal{M}$ is always defined. Now we give a formal definition of the correctness of parameter passing.

5.3 Definition (Correctness of parameter passing)

We assume the following parameter passing diagram.



Standard parameter passing is *correct* if PSPEC is freely extendible and if for every passing morphism h: $SPEC_p \rightarrow SPEC'$ the following holds:

$$SEM(PSPEC_h(SPEC')) = \cup_{\mathcal{F} \in SEM(PSPEC)} \mathcal{F}_h(SEM(SPEC')) \qquad (*)$$

where $\mathcal{F}_h(SEM(SPEC')) = \{ \mathcal{F}_h(\mathcal{M}) : \mathcal{M} \in SEM(SPEC') \text{ and } V_h(\mathcal{M}) \in dom(\mathcal{F}) \}$. ■

A careful look at the definitions reveals, that the right to left inclusion of the set equation (*) in 5.3 is generally true. However, the reverse does not hold. There exist freely extendible specifications that do not have correct parameter passing. Essentially, the reason why a parameterized specification PSPEC might fail to have correct parameter passing is the following phenomenon: It might happen that for a certain h: $SPEC_p \rightarrow SPEC'$ there exists a generated model $\mathcal{B}$ of ASPEC, the SIG' reduct of which is not generated (see [Re 91] for examples). Fortunately, this is actually the only phenomenon hindering the correctness of parameter passing. The following definition is designed to prevent it.

5.4 Definition (Strict persistency)

Let PSPEC be a parameterized specification with signature $SIG = SIG_p \cup (S, OP)$ and parameter signature $SIG_p = (S_p, OP_p)$. PSPEC is *strictly persistent*, if for all target models $\mathcal{A} \in Tar(PSPEC)$, for all $s \in S_p$, all terms $t(x_1, .., x_n) \in T_{OP \cup OP_p, s}(X_p)$ and all valuations $s : X_p \rightarrow \mathcal{A}$ there exists a term $t' \in T_{OP_p, s}(\{x_1, .., x_n\})$ with: $t_{s, \mathcal{A}} = t'_{s, \mathcal{A}}$. ■

Under weak restrictions of SIG_p, strict persistency plus free extendibility characterize the specifications with correct parameter passing.

5.5 Theorem (Characterization of correct parameter passing)

Let $PSPEC = (SPEC_p, SPEC)$ be a parameterized specification with signature $SIG = SIG_p \cup (S, OP)$. The parameter signature $SIG_p = (S_p, OP_p)$ is required to provide at least one term $t \in T_{OP_p, s}$ for all $s \in S_p$. Then the following holds:

Standard parameter passing is *correct* for PSPEC iff PSPEC is *strictly persistent* and *freely extendible*. ■

5.6 Remark (Strict persistency and related work)

In theorem 5.5 the "if-part" of the equivalence is relevant for practical applications. However, the restriction on SIG_p in 5.5 is only needed for the "only-if-part". Therefore, the restriction is irrelevant for practical applications. It can be dropped completely, if SIG_p has only a single parameter sort.

In order to compare result 5.5 with the corresponding one for initial semantics ([EM 85]), we assume (E_p, E) to be a freely extendible, parameterized equational specification. Then all possibly non-isomorphic interpretations of $SEM((E_p, E))$ are defined for all parameter algebras and are persistent. In the initial approach, persistency of the free functor already implies the correctness of parameter passing, whereas we need the additional requirement that (E_p, E) is strictly persistent. This is because persistent free functors additionally guarantee, that the reduct of the image of a generated actual parameter algebra is always generated. This is exactly what strict persistency guarantees in our setting. The definitions of strict persistency and free extendibility have also some similarity to the definitions of *sufficient completeness* and *consistency* in [Ga 83] for equational specifications. Actually, the corresponding result for (the standard version of) parameter passing in [Ga 83] for equational specifications can be obtained as a corollary of ours. The approach in [NO 87] is based on functorial semantics and considers Horn specifications with boolean constraints. Here we consider loose semantics and full first-order logic.

The following syntactic criterion is useful to establish strict persistency. It states that PSPEC is strictly persistent if the target operations can be eliminated from terms of a parameter sort.

5.7 Theorem (Criterion for strict persistency)

PSPEC = $(SPEC_p, SPEC)$ with signature $SIG = (S \cup S_p, OP \cup OP_p)$ is *strictly persistent*, if for all $s \in S_p$, all terms $t(x_1, .., x_n) \in T_{OP \cup OP_p,s}(X_p)$ there exist terms $t_1', .., t_k' \in T_{OP_p,s}(\{x_1, .., x_n\})$ such that $SPEC \cup IND(SIG) \models \bigvee_{i=1,..,k} t = t_i'$. ■

In 5.7 it is not necessary that $SPEC \cup IND(SIG)$ determines a single t' for t. It may leave unspecified which of the terms t_i' is equal to t. In our sample specification Sets_over_Partial_order, there are no terms of a parameter sort at all. Therefore, 5.7 is vacuously true, and hence Sets_over_Partial_order is strictly persistent.

Now we turn our attention to the interaction of parameter passing and the correctness of the involved specifications. Correct parameter passing, or equivalently strict persistency plus free extendibility, and correctness of a specification are independent notions: There are strictly persistent and freely extendible specifications that are not correct, and conversely, there exist freely extendible specifications that are correct for an interpretation but not strictly persistent. However, strict persistency and free extendibility are important to ensure that actualization preserves the correctness of the involved specifications.

5.8 Theorem (Actualization and correct specifications)

If PSPEC = ($SPEC_p$, SPEC) is strictly persistent and freely extendible, then for all actual parameter specifications SPEC', and all passing morphisms h: $SPEC_p \rightarrow$ SPEC' the following holds:

PSPEC correct for $\mathcal{F}$, and SPEC' correct for $\mathcal{M}$ $\Rightarrow$ $PSPEC_h$(SPEC') correct for $\mathcal{F}_h(\mathcal{M})$ ■

Theorem 5.8 suggests to look at strictly persistent and freely extendible specifications, in order to guarantee correctness preserving actualization. Strict persistency can be checked by criterion 5.7. A specification PSPEC can be shown to be freely extendible by giving a concrete function $\mathcal{F}$ with $dom(\mathcal{F}) = Alg(SPEC_p)$ which satisfies PSPEC. This can be done with our sample specification Sets_over_Partial_order. Consequently, actualization of Sets_over_Partial_order is correctness preserving. Under certain restrictions there is also a syntactic criterion for free extendibility.

5.9 Theorem (Criterion for free extendibility)

PSPEC with signature SIG and parameter signature SIG_p is freely extendible if the following conditions hold:

1) PSPEC strictly persistent

2) SPEC $\subseteq \forall L(SIG, X)$

3) SIG provides a quasi-ground term for each target sort

4) for all $\varphi \in \forall L(SIG_p, X_p)$: SPEC $\vdash \varphi \Rightarrow SPEC_p \vdash \varphi$.

($\vdash$ denotes derivability in first-order logic) ■

References

[BDPPW 79] Broy, M., Dosch, H., Partsch, H., Pepper, P., Wirsing, M., Existential Quantifiers in Abstract Data Types, Proc. 6th ICALP, Graz 1979, Springer LNCS 71

[BG 80] Burstall, R.M., Goguen, J. A., The semantics of CLEAR, a specification language, in: D. Bjorner, ed., Proc. Advanced Course on Abstract Software Specifications, Springer LNCS 86, 1980

[Br 91] Broy, Facchi, Grosu, Hettler, Hussmann, Nazareth, Regensburger, Stolen, The Requirement and Design Specification Language SPECTRUM, technical report TUM-I9140, TU Munich, 1991

[CIP 87] CIP-Group, The Munich Project CIP, Vol. 2, The Program Transformation System CIP-S, Springer LNCS 292 (1987)

[Eh 82] Ehrich, H.-D., On the theory of specification, implementation, and parameterization, of abstract data types, JACM 29(1), 1982

[EKTWW 80] Ehrig, Kreowski, Thatcher, Wagner, Wright, Parameterized data types in algebraic specification languages, Proc. Int. Coll. on Automata, Languages and Programming, Springer LNCS 85, 1980

[EM 85] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification 1, Equations and Initial Semantics, EATCS Monographs on Theoretical Computer Science, Vol. 6, Springer 1985

[EM 90] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification 2, Module Specifications and Constraints, EATCS Monographs on Theoretical Computer Science, Vol. 21, Springer 1990

[Ga 83] Ganzinger, H., Parameterized Specifications: Parameter Passing and Implementation with Respect to Observability, ACM TOPLAS, Vol. 5, No. 3, July 1983

[Gau 92] Gaudel, M.-C., Structuring and Modularizing Algebraic Specifications: the PLUSS specification language, evolutions and perspectives, Finkel, Jantzen (eds.), STACS 92, Springer LNCS 577, 1992

[GGM 76] Giarratana, V., Gimona, F., Montanari, U., Observability Concepts in Abstract Data Type Specifications, 5th Symposium Math. Foundations of Computer Science (1976), Springer LNCS 45

[GTW 78] Goguen, J., Thatcher, J., Wagner, E., An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, Current Trends in Programming Methodology IV, Yeh, R. (Ed.), Prentice-Hall, Englewood Cliffs, 1978

[Gu 76] Guttag, J.V., Abstract Data Types and The Development of Data Structures, Supplement to Proc. Conference on Data Abstraction, Definition, and Structure, SIGPLAN Notices 8 (1976)

[HR 80] Hornung, G., Raulefs, P., Terminal Algebra Semantics and Retractions for Abstract Data Types, DeBakker, van Leuwen, 7th ICALP, Springer LNCS 85, pp. 310-323 (1980)

[Ka 83] Kamin, S., Final Data Types and Their Specification, ACM TOPLAS 5,1 1983

[LZ 74] Liskov, B.H., Zilles, S.N., Programming with Abstract Data Types, SIGPLAN Notices 6 (1974)

[NO 87] Navarro M., Orejas F., Parameterized Horn Clause Specifications: Proof Theory and Correctness, TAPSOFT 87, Springer LNCS 249, 1987

[Pa 85] Padawitz, P., Parameter Preserving Data Type Specifications, J. of Computer and System Sciences 34

[Pa 90] Padawitz, P., Horn Logic and Rewriting for Functional and Logic Program Design, Universität Passau, Fakultät für Mathematik und Informatik, MIP-9002, März 1990

[Re 91] Reif, W., Correctness of Specifications and Generic Modules, Dissertation, Univ. of Karlsruhe, 1991

[Re 92] Reif, W., Correctness of Full First-order Specifications, Proc. International Conference on Software Engineering and Knowledge Engineering, Capri, Italy, IEEE press, 1992

[SST 92] Sanella, D., Sokolowski, S., Tarlecki, A., Toward formal development of programs from algebraic specifi cations: parametrisation revisited, report ECS-LFCS-92-222, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992

[ST 88] Sanella, D.,, Tarlecki, A., Specifications in an arbitrary institution, Inform. a. Comp. 76, 1988

[SW 83] Sanella D., Wirsing M., A kernel language for algebraic specification and implementation, Proc. Int. Conf. on Foundations of Conputation Theory, Borgholm, Springer LNCS 158, 1983

[TWW 82] Thatcher, J., Wagner, E., Wright, J., Data Type Specification: Parametrization and the Power of Specification Techniques, ACM TOPLAS 4,4 1982

[Wa 79] Wand, M., Final Algebra Semantics and Data Type Extensions, Journal of Computer and System Sciences 19, 1, 1979

[WB 82] Wirsing, M., Broy, M., An Analysis of Semantic Models for Algebraic Specifications, Theoretical Foundations of Programming Methodology, Broy, Schmidt (eds.), Nato ASIS, C91, Reidel, 1982

[Wi 86] Wirsing, M., Structured algebraic specifications: a kernel language, Theoretical Computer Science 42, 1986

[Wi 90] Wirsing, M., Algebraic Specification, Handbook of Theoretical Computer Science, J. van Leeuwen (ed.), volume B, Elsevier, 1990

[WPPDB 83] Wirsing, M., Pepper, P., Partsch, H., Dosch, W., Broy, M., On Hierarchies of Abstract Data Types, Acta Informatica 20 (1983)