KES.U.96.8

Kestrel Institute

Refinement of Parameterized Algebraic Specifications

by

Yellamraju V. Srinivas

September 1996
Revised January 1997

To appear in
Proceedings of the *IFIP TC2 Working Conference on Algorithmic Languages and Calculi* (17-22 February 1997, Le Bischenberg, France).

KESTREL INSTITUTE • 3260 HILLVIEW AVENUE • PALO ALTO, CA 94304 • (415) 493-6871

Refinement of
Parameterized Algebraic Specifications

Yellamraju V. Srinivas
Kestrel Institute
3260 Hillview Avenue, Palo Alto, CA 94304, USA
Email: srinivas@kestrel.edu, Tel: (415)493-6871, Fax: (415)424-1807

Abstract
A refinement relation for parameterized algebraic specifications is introduced in which the body specification is refined covariantly (i.e., specialized) while the parameter specification is refined contravariantly (i.e., generalized). This refinement is similar to the subtyping relation between function types. Assuming a few relatively weak axioms about parameterized specifications, refinement is shown to satisfy the expected properties for software development in-the-large: independent refinement of body and parameter, and closure under sequential composition, instantiation and a suitable form of 'horizontal' composition.

Keywords
Parameterized specifications, mixed-variance refinement, compositionality

1 INTRODUCTION

The primary subject of this paper is a general notion of refinement for parameterized specifications. Many notions of refinement have been previously proposed in the literature (Ehrig & Kreowski 1982, Ganzinger 1983, Goguen & Meseguer 1982, Sannella & Tarlecki 1988, Sannella & Wirsing 1982, Ehrig & Mahr 1990, Ehrig & Große-Rhode 1994). In many of these, the parameter is not refined; in some, the parameter is refined, either in the same direction as the body (covariant refinement) or in the opposite direction (mixed-variance refinement). We investigate the properties of a somewhat more flexible mixed-variance refinement relation.

Consider, for example, a specification for sorting, which is parameterized on a total order. After refinement, we will have a program, say mergesort. Such a program is typically parameterized, not on a total order, but the *signature* of a total order, i.e., a type and a binary relation; this is because most programming languages cannot express the requirement that the relation satisfy the axioms of a total order. Indeed, 'merge' and 'mergesort' may be defined as operations on sequences given any binary relation; although these functions will always produce a result, the results may not be meaningful. However, when 'merge' is given sorted sequences and a total order as arguments, it will produce a sorted sequence. To show the correctness of such a refinement, we need a refinement relation between parameterized specifications in which the parameter can be changed.

The refinement relation that is described in this paper is similar to the subtyping relation between function types: the body is specialized while the parameter is generalized. In our example above, mergesort is a valid refinement of the sorting specification, if after instantiation by a total order, mergesort is a refinement of sorting.

Our notion of refinement is mostly independent of the semantics of parameterized specifications. We show that, assuming a few relatively weak axioms about parameterized specifications, refinement satisfies the expected properties for software development in-the-large: independent refinement of body and parameter, and closure under sequential composition, instantiation and a suitable form of horizontal composition. The existence of such a refinement calculus is especially pleasing, given that several semantics have been proposed for parameterized specifications (Ehrig *et al.* 1981, Bidoit 1987, Wirsing 1986).

2 SPECIFICATIONS, MORPHISMS, INTERPRETATIONS

We will assume a category of specifications and morphisms in which finite colimits exist. The constructions and calculations in this paper are based on this category and sub-categories consisting of morphisms with special properties. We only discuss models briefly; for this purpose, we assume a (contravariant) functor *Mod* which assigns to each specification a category of models and to each morphism $\sigma: A \rightarrow B$ a functor $\_|_\sigma: Mod(B) \rightarrow Mod(A)$ called *reduct* functor.

Example specifications use a specific logic, namely the logic of SPECWARE, a formal software development system currently being developed at Kestrel (Srinivas & Jüllig 1995). A specification in SPECWARE is a finite presentation of a theory in higher-order logic, i.e., a finite collection of sorts, operations, and axioms (Lambek & Scott 1986). A specification morphism in SPECWARE is a finite map from sorts to sorts and operations to operations such that the ranks of operations are preserved and axioms translate to theorems.

2.1 Refinement by Interpretations

The general notion of refinement is that a specification B refines a specification A if there is a construction which produces models of A from models of B (Sannella & Tarlecki 1988). Specification morphisms serve this purpose because associated with every morphism $\sigma: A \rightarrow B$ there is a reduct functor $\_|_\sigma$ which produces models of A from models of B. Morphisms, however, are too weak to represent refinements which normally occur during software development. So, we use a more general notion, *interpretations*, which are specification morphisms from the source specification to a definitional extension (see below) of the target specification. Methodologically, we believe that two different kinds of arrows are useful in software developement: a simple arrow (morphism) for structuring specifications, and a more flexible arrow (interpretation) for refining specifications.

2.2 Definitional Extensions

Definitional extensions are a subclass of morphisms, those which extend a specification with only defined sorts and operations, while otherwise leaving the specification unchanged. Model theoretically, the reduct functor corresponding to a definitional extension is an isomorphism.

Different logics may have different notions of definability. For example, in Specware, to define an operation $f\colon A \to B$ in a specification S, we must exhibit a formula φ of S which denotes the graph of f. The formula φ must contain exactly two free variables $x\colon A$, $y\colon B$ and must be provably functional in S:

$$S \vdash \forall x\colon A \cdot \exists! y\colon B \cdot \varphi(x,y).$$

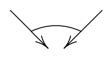
S can then be extended with the operation f and its defining axiom

$$\forall x\colon A,\ y\colon B \cdot f(x) = y \iff \varphi(x,y).$$

2.3 Axioms for definitional extensions

Rather than formally defining definitional extensions, we will assume the following axioms which will be sufficient for our constructions and calculations:

1. An identity morphism is a definitional extension.
2. A definitional extension is an epimorphism.
3. Definitional extensions are closed under composition.
4. Pushouts preserve definitional extensions.

Figure 1 renders these axioms pictorially. Definitional extensions are indicated in diagrams* by $\relbar d \!\!\rightarrow$ and pushouts by  .

*All diagrams are assumed to be commutative unless stated otherwise. Labels on arrow shafts (e.g., $\relbar d \!\!\rightarrow$, $\relbar p \!\!\rightarrow$) are properties of arrows, whereas labels above or below (e.g., $\xrightarrow{x}$) are names of arrows.

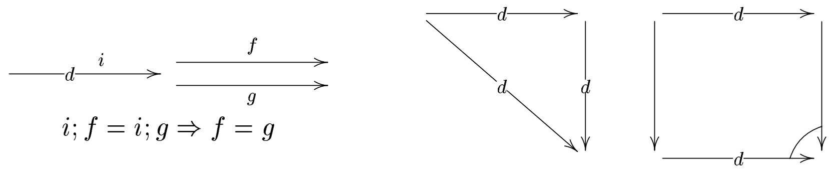


Figure 1 Axioms for definitional extensions.

Alternatively, these axioms will be theorems if we assume that the model functor *Mod* maps pushouts to pullbacks and if we define a definitional extension to be a morphism whose image via *Mod* is an isomorphism.

2.4 Interpretations

DEFINITION 2.1: *Interpretation.* An *interpretation* $\rho: A \Rightarrow B$ from a specification A (called *domain* or *source*) to a specification B (called *codomain* or *target*) is a pair of morphisms $A \longrightarrow B_+ \xleftarrow{d} B$ with common codomain B_+ (called *mediating* specification or simply *mediator*), such that the target-to-mediator morphism is a definitional extension. □

Interpretations are a suitable notion of refinement because models of the source specification can be constructed from models of the target specification by first expanding them along the definitional extension and then taking reducts.

Specifications and interpretations form a category with composition defined as in Figure 2. We use the facts that definitional extensions are closed under composition and are preserved by pushouts. Equality of interpretations is defined by reducing it to equality of morphisms using the construction in Figure 3. In other words, the mediator specification may contain extra defined sorts and operations which are not in the image of the source-to-mediator morphism.

The category of specifications and interpretations is finitely cocomplete if the category of specifications and morphisms is. Both categories have the same initial object. Pushouts of interpretations are constructed as shown in Figure 4 (note that a morphism may be construed as an interpretation, by pairing it with an identity definitional extension). A proof of universality is given in the appendix.

$$\begin{tikzcd}
 & & A \arrow[dl] \arrow[dd] \\
 & B_+ \arrow[dr] & \\
B \arrow[ur, "d"] \arrow[dr] & & {} \\
 & C_+ \arrow[ur, "d"] & \\
 & & C \arrow[ul, "d"] \arrow[uu, "d"]
\end{tikzcd}
\qquad
\begin{tikzcd}
 & A \arrow[dl, Rightarrow] \arrow[dd, Rightarrow] \\
B \arrow[dr, Rightarrow] & \\
 & C
\end{tikzcd}$$

Figure 2 Composition of interpretations.

$$\begin{tikzcd}
 & & B_+^1 \arrow[dl, "d"] & \\
A \arrow[urr] \arrow[drr] \arrow[r, shift left] \arrow[r, shift right] \arrow[r, phantom, "=?"] & B_+^{12} & & B \arrow[ul, "d"] \arrow[dl, "d"] \\
 & & B_+^2 \arrow[ul, "d"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, Rightarrow, shift left=2] \arrow[r, Rightarrow, shift right=2] \arrow[r, phantom, "=?"] & B
\end{tikzcd}$$

Figure 3 Equality of interpretations.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & C_+ \arrow[ddr] & C \arrow[l, "d"'] \arrow[dd] \\
B_+ \arrow[drr] & & \\
B \arrow[u, "d"] \arrow[rr] & & D
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, Rightarrow] \arrow[d, Rightarrow] & C \arrow[d] \\
B \arrow[r] & D
\end{tikzcd}$$

Figure 4 Pushout of interpretations.

Note. Diagrams on the left are in the category of specifications and morphisms; those on the right are in the category of specifications and interpretations.

3 PARAMETERIZED SPECIFICATIONS

A parameterized specification is a specification morphism with special properties. The domain is called the *parameter* and the codomain the *body* of the parameterized specification. The morphism shows how the parameter is a part-of (or fits into) the body.

The parameter represents the variable part and may be *instantiated* by providing a morphism into an *instantiating* (or *actual*) specification. The result of this instantiation is obtained by computing the pushout of the parameterizing morphism and the instantiating morphism. Since the parameter is a specification, it can contain axioms which will impose semantic constraints on any specification that is used to replace the parameter.

This approach to parameterization is similar to that adopted in CLEAR (Burstall & Goguen 1977). Semantically, a parameterized specification represents some flavor of function mapping models of the parameter to models of the body. For example, with intial semantics for specifications, a parameterized specification can denote a free functor (Ehrig *et al.* 1981). Or with loose semantics, it may denote a class of functors (Bidoit 1987, Ehrig & Große-Rhode 1994).

Rather than assume a particular semantics, we will assume some axioms about parameterized specifications, and use these axioms in defining and reasoning about refinement. In the rest of the paper, the phrase 'parameterized specification' will be abbreviated by 'pspec'.

3.1 Axioms for pspecs

1. An identity morphism is a pspec.
2. Pspecs are closed under composition.
3. A definitional extension is a pspec.
4. Pushouts preserve pspecs.

Pspecs are indicated in diagrams by $\xrightarrow{p}$.

3.2 Instantiation with interpretations

While reasoning with refinements of pspecs, it is convenient to be able to instantiate a pspec with an interpretation rather than a morphism, as shown in Figure 5. Syntactically, this means that some of the vocabulary in the parameter is not already present in the instantiating specification, so it needs to be extended before the parameter can be mapped via a morphism. Semantically, an interpretation represents a restriction of the model class, so instantiation with an interpretation is similar to that by a morphism.

$$\begin{tikzcd}
S \arrow[d] & P \arrow[l, "p"'] \arrow[d] \\
S(A) \arrow[ur, phantom, "\ulcorner", very near start] & A_+ \arrow[l, "p"'] \\
& A \arrow[u, "d"] \arrow[ul, "p"]
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[d] & P \arrow[l, "p"'] \arrow[d, Rightarrow] \\
S(A) \arrow[ur, phantom, "\ulcorner", very near start] & A \arrow[l, "p"']
\end{tikzcd}$$

Category of morphisms   Category of interpretations

Figure 5 Instantiation of pspecs with interpretations.

Notation. The instantiation of a pspec s: $P \xrightarrow{p} S$ with an interpretation $a{:}\,P \Rightarrow A$ will be written as $s(a)$ and will ambiguously denote either the pushout specification $S(A)$ or the resulting pspec $A \xrightarrow{p} S(A)$. The notation $S(A)$, with S and A being specifications rather than morphisms, will sometimes be used to denote the pushout specification when the parameterization and instantiation morphisms are clear from the context.

Instantiation with interpretations emphasizes the asymmetry between the parameterization and instantiation morphisms: the former is like a 'part-of' relation while the latter is like an 'is-a' relation. Yet instantiations behave like pushouts; indeed, they *are* pushouts in the category of specifications and interpretations. We record below a well known property of pushouts that we will subsequently use in our constructions.

LEMMA 3.1: *Cascaded instantiations.* In the commuting diagram below,

1. if both the squares are instantiations, then the outer rectangle is also one, and
2. if the left square and the outer rectangle are instantiations, then the right square is also one.

$$\begin{tikzcd}
P \arrow[r, Rightarrow] \arrow[d, "p"] & A \arrow[r, Rightarrow] \arrow[d, "p"] & B \arrow[d, "p"] \\
S \arrow[r] & S(A) \arrow[r] \arrow[ul, phantom, "\lrcorner", very near start] & S(B) \arrow[ul, phantom, "\lrcorner", very near start]
\end{tikzcd}$$

□

spec ELEM
 sort ELEM
end-spec

spec SEQ
 sorts ELEM, SEQ
 op $[\,]$: SEQ
 op $\_ : \_$: ELEM, SEQ $\to$ SEQ
 axiom $a{:}\, S \neq [\,]$
 axiom $a{:}\, S = b{:}\, T \Rightarrow S = T$
 axiom $\{\, [\,], \_ : \_ \,\}$ *construct* SEQ
end-spec

spec TOT
 sort ELEM
 op $\_ \leq \_$: ELEM, ELEM $\to$ BOOL
 axiom $x \leq x$
 axiom $x \leq y \wedge y \leq x \Rightarrow x = y$
 axiom $x \leq y \wedge y \leq z \Rightarrow x \leq z$
 axiom $x \leq y \vee y \leq x$ —*totality*
end-spec

spec SORTING
 import SEQ(TOT)
 op ordered?: SEQ $\to$ BOOL
 definition of ordered? *is*
  ordered?($[\,]$)
  ordered?($a{:}\,[\,]$)
  ordered?($a{:}\,b{:}\,S$)
   iff $a \leq b \wedge$ ordered?(S)
 end-definition
 op perm?: SEQ, SEQ $\to$ BOOL
   —*definition omitted*
 op sort: SEQ $\to$ SEQ
 axiom sort(S) $= T$
    iff perm?(S, T) $\wedge$ ordered(T)
end-spec

$$\begin{tikzcd}
 & \text{ELEM} \arrow[dl, "p"'] \arrow[dr] & \\
\text{SEQ} \arrow[dr] & & \text{TOT} \arrow[dl, "p"] \arrow[ddl, "p"] \\
 & \text{SEQ(TOT)} \arrow[d, "d"] & \\
 & \text{SORTING} &
\end{tikzcd}$$

Figure 6 A parameterized specification for sorting.

EXAMPLE 3.1: *Parameterized specifications.* Figure 6 shows two parameterized specifications, one for sequences and one for sorting.

The specification for sequences, SEQ, is parameterized on the type of the elements. The parameter specification ELEM postulates a sort for the elements but leaves the sort unconstrained. The specification SEQ contains two operations for constructing sequences, two axioms which characterize equality of sequences, and an induction axiom.

To specify sorting, we need sequences in which the elements have a total order. Such sequences are obtained by instantiating SEQ with the specification for a total order, TOT. The specification SORTING extends this instantiation with predicates used to define sorting. SORTING is itself parameterized with TOT. □

4 REFINEMENT OF PARAMETERIZED SPECIFICATIONS

As noted in the introduction, we will consider refinements in which the body is refined covariantly while the parameter is refined contravariantly. Semantically, a pspec behaves like a function from models of the parameter to models of the body. Such a function can be refined to, or realized by, another function which has a larger domain. Hence, in a pspec refinement, the parameter is relaxed or refined contravariantly. Since the target pspec has a less constrained parameter, it must first be instantiated with the source parameter. With the parameters matched, the bodies may then be related by an interpretation.

Definition 4.1: *Pspec interpretation.* A ***pspec interpretation*** $s \overset{p}{\Longrightarrow} s'$ from a pspec s: $P \xrightarrow{p} S$ to a pspec s': $P' \xrightarrow{p} S'$ is a pair of interpretations, $\langle \pi : P' \Rightarrow P, \beta : S \Rightarrow s'(\pi) \rangle$, where π refines the parameter contravariantly and β refines the body of the source pspec into the instantiation of the target pspec with π, such that $s;\beta = s'(\pi)$. □

Figure 7 renders this definition pictorially, and also shows two special cases of pspec interpretations in which either the body or the parameter is refined via the identity. Every pspec interpretation can be decomposed into a pair consisting of a body refinement followed by a parameter refinement. This decomposition will be used subsequently to build up other constructions with pspec interpretations.

$$\begin{tikzcd}
s \arrow[d, Rightarrow, "\langle \pi,\beta \rangle"', "p"] \\
s'
\end{tikzcd}
\qquad
\begin{tikzcd}
S & & P \arrow[ll, "s"', "p"] \arrow[dll, "p", "s'(\pi)"'] \\
S'(P) \arrow[from=u, Rightarrow, "\beta"] & & \\
S' \arrow[u] & & P' \arrow[ll, "s'", "p"'] \arrow[uu, Rightarrow, "\pi"']
\end{tikzcd}$$

Special Cases

Body refinement (covariant) — Parameter refinement (contravariant)

$$\begin{tikzcd}
S \arrow[d, Rightarrow] & P \arrow[l, "p"'] \arrow[dl, "p"] \\
S' &
\end{tikzcd}
\qquad
\begin{tikzcd}
T'(Q) & Q \arrow[l, "p"'] \\
T' \arrow[u] & Q' \arrow[l, "p"'] \arrow[u, Rightarrow]
\end{tikzcd}$$

Figure 7 Refinement of pspecs.

```
spec TOT-SIG
 sort ELEM
 op _ ≤ _: ELEM, ELEM → BOOL
end-spec

spec MSORT
 import SEQ(TOT-SIG)
 op merge: SEQ, SEQ → SEQ
 definition of merge is
   merge([], T) = T
   merge(S, []) = S
   merge(a: S, b: T) =
   if a ≤ b then a: merge(S, b: T)
    else b: merge(a: S, T)
 end-definition
 op split: SEQ → SEQ, SEQ
        —definition omitted
 op msort: SEQ → SEQ
 definition of msort is
   msort(S) =
   if S = [] then []
    else let ⟨S_1, S_2⟩ = split(S) in
    merge(msort(S_1), msort(S_2))
 end-definition
end-spec
```

$$\begin{tikzcd}
 & \textsc{Elem} \arrow[dl, "p"'] \arrow[dr] & \\
\textsc{Seq} \arrow[dr] & & \textsc{Tot-Sig} \arrow[dl, "p"'] \arrow[ddl, bend left, "p"] \\
 & \textsc{Seq}(\textsc{Tot-Sig}) \arrow[d, "d"] & \\
 & \textrm{M}\textsc{Sort} &
\end{tikzcd}$$

$$\begin{tikzcd}
\textsc{Sorting} \arrow[d] & \textsc{Tot} \arrow[l, "p"'] \arrow[dl, "p"] \\
\textrm{M}\textsc{Sort}(\textsc{Tot}) & \\
\textrm{M}\textsc{Sort} \arrow[u] & \textsc{Tot-Sig} \arrow[l, "p"'] \arrow[uu]
\end{tikzcd}$$

Figure 8 Pspec interpretation for sorting.

EXAMPLE 4.2: *Pspec interpretation.* Figure 8 shows a pspec interpretation from the pspec for sorting to a mergesort specification parameterized on the signature of a total order, i.e., a sort together with a binary relation.

The top right part of the figure shows the construction of the target pspec MSORT: SEQ is instantiated with the signature of a total order, TOT-SIG, and then extended with definitions of split, merge and mergesort. Note that this mergesort operation, although a total function, is not guaranteed to produce a sequence which is ordered, because the $\leq$ operation may not be a total order. Yet, this is the mergesort operation that is implemented in most programming languages.

The bottom right part of the figure shows a pspec interpretation. The target parameter TOT-SIG is refined into the source parameter TOT via an inclusion morphism. Once the two parameters are connected, the sorting specification can be refined into the instantiation of the target pspec, MSORT(TOT). The latter specification has enough axioms about $\leq$ to show the correctness of the refinement. □

5 INSTANTIATION OF PSPEC INTERPRETATIONS

Instantiating a pspec yields another pspec. This process can be lifted to pspec interpretations: any interpretation of the original pspec yields an interpretation of the instantiated pspec.

Figure 9 shows the effect of instantiation on a pspec interpretation. In the left half of the figure, we only consider a body refinement. Universality of instantiation yields a unique interpretation from the instantiation of the source pspec, $S(A)$, to the instantiation of the target pspec, $S'(A)$. In the right half of the figure, this is combined with parameter refinement. Using lemma 3.1, composing the parameter refinement with the instantiation yields another parameter refinement.

$$\begin{tikzcd}
& P \arrow[dl, "p"'] \arrow[dddl, "p"] \arrow[drr, Rightarrow] & & \\
S \arrow[dd, Rightarrow] \arrow[drr] & & & A \arrow[dl, "p"'] \arrow[dddl, "p"] \\
& & S(A) \arrow[dd, Rightarrow, dashed] & \\
S' \arrow[drr] & & & \\
& & S'(A) &
\end{tikzcd}
\qquad
\begin{tikzcd}
& & P \arrow[dll, "p"'] \arrow[ddll, "p"] \arrow[drrr, Rightarrow] & & & \\
S \arrow[d, Rightarrow] \arrow[drrrr] & & & & & A \arrow[dl, "p"'] \arrow[ddl, "p"] \\
S'(P) \arrow[drrrr] & & P' \arrow[uu, Rightarrow] \arrow[dll, "p"'] \arrow[drrr, "\mathrm{id}"] & & S(A) \arrow[d, Rightarrow] & \\
S' \arrow[u] \arrow[drrrr, "\mathrm{id}"] & & & & S'(A) & P' \arrow[uu, Rightarrow] \arrow[dl, "p"'] \\
& & & & S' \arrow[u] &
\end{tikzcd}$$

Figure 9 Instantiation of pspec interpretations.

5.1 Independent refinement of pspec and actual

The main reason for parameterizing a specification is the possibility of independently refining the parameterized specification and the instantiating (or actual) specification, and later combining the two refinements, with the parameter capturing the interface between the two. In Figure 10, we show how to parallely compose a pspec interpretation and an interpretation of the actual to obtain an interpretation of the instantiation. The body interpretation is first transferred along the instantiation (as in Figure 9). The resulting pspec, $A \xrightarrow{p} S'(A)$, is then instantiated using the interpretation of the actual. The result, as shown in the right half of the figure, is an interpretation from

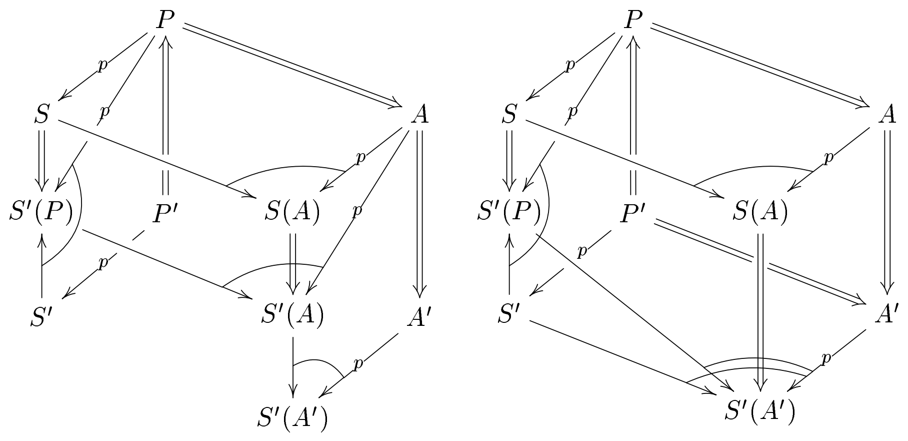


Figure 10 Parallel composition of pspec and actual interpretations.

the instantiation of the source pspec with the actual to the instantiation of the target pspec with the refinement of the actual.

The information in the parallel composition of a pspec interpretation and an actual interpretation is completely contained in the body interpretation of the pspec and the interpretation of the actual. In the right half of Figure 10, just as $S(A)$ is covered by the pushout morphisms $S \to S(A) \leftarrow A$ and $S'(A')$ is covered by the pushout morphisms $S' \to S'(A') \leftarrow A'$, so is the parallely composed interpretation $S(A) \Rightarrow S'(A')$ covered by the morphism pair from $S \Rightarrow S'(P)$ and the morphism pair from $A \Rightarrow A'$.

5.2 Instantiation with sharing

In the instantiations we have considered until now, the only part shared by the body and the actual is the parameter. Sometimes, the body and actual share more: e.g., both may import a standard specification such as that of integers. In this case, a simple instantiation will create two copies of the integer specification in the instantiated specification. We can avoid this by explicitly indicating the extra sharing, via morphisms, and taking the colimit of the larger diagram. In the top part of Figure 11, we explicitly indicate extra sharing via two morphisms from the shared specification C. The effect of this sharing is a (coequalizing) morphism from the unshared instantiation $S(A)$ to $S(A)_C$ in which the two copies of C are collapsed.

Now, to parallely compose the pspec and actual refinements, we have to ensure that the shared part is refined similarly in both the body and the actual. This compatibility requirement is satisfied by exhibiting morphisms[*]

[*]A morphism of interpretations is a pair of specification morphisms, one between the do-

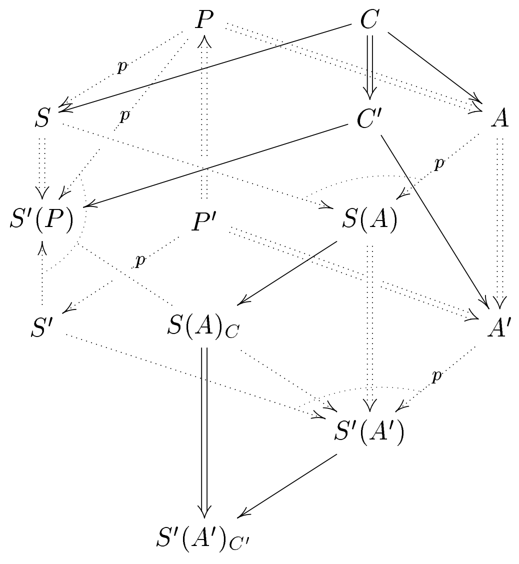


Figure 11 Refinement of shared instantiation.

from the interpretation of the shared part to the interpretations of the body and actual. Then, the target also becomes a shared instantiation, $S'(A')_{C'}$, and we get an interpretation from $S(A)_C$ by universality. The construction is similar to the parallel composition of a diagram refinement, as described in (Srinivas & Jüllig 1995).

6 A CATEGORY OF PSPEC INTERPRETATIONS

It is easy to see that body interpretations and parameter interpretations are each closed under sequential composition. Figure 12 shows the sequential, or vertical, composition of (general) pspec interpretations. The main part of the construction is conversion of a parameter interpretation followed by a body interpretation into a body interpretation followed by a parameter interpretation. This is done by instantiating the second body interpretation with the first parameter interpretation (as in Figure 9). Thus pspecs and pspec interpretations form a category, with the obvious identity arrows, and associativity following from the associativity property for interpretations and pushouts.

To define equality of pspec interpretations, we have to handle isomorphisms between instantiations of the target pspec. Given two pspecs, s: $P \xrightarrow{p} S$

mains and one between the codomains, such that the resulting square commutes in the category of specifications and interpretations.

and $s'\colon P' \xrightarrow{p} S'$, two pspecs interpretations, $\langle \pi\colon P' \Rightarrow P, \beta\colon S \Rightarrow s'(\pi)\rangle$ and $\langle \pi'\colon P' \Rightarrow P, \beta'\colon S \Rightarrow s'(\pi')\rangle$, are equal if $\pi = \pi'$ and $\beta = \beta'; i$ where $i\colon\ s'(\pi') \xrightarrow{\sim} s'(\pi)$ is the isomorphism between the instantiations of the target pspec with the two (equal) parameter interpretations.

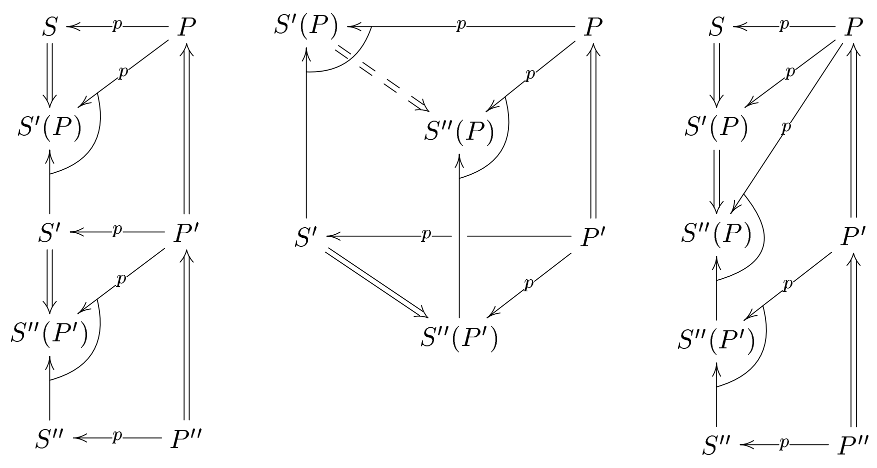


Figure 12 Sequential (vertical) composition of pspec interpretations.

7 HORIZONTAL COMPOSITION OF PSPEC INTERPRETATIONS

Consider an instantiation of a pspec by the body of another pspec. The result is another pspec parameterized on the parameter of the second pspec. A typical example is Set(Seq(X)). Any instantiation of such a 'iterated' pspec satisfies an associativity law, e.g., Set(Seq(Nat)) = (Set(Seq))(Nat); the former is cascaded instantiation while the latter is an instantiation of an iterated pspec. This associativity law is depicted in Figure 13 and follows from the properties of pspecs and instantiations.

We now consider the construction of an interpretation for an iterated pspec from interpretations of the component pspecs. This construction is always possible for body refinements, as shown in the top half of Figure 14. The source iterated pspec $Q \xrightarrow{p} T \xrightarrow{p} S(T)$ is obtained by instantiating the outer pspec $P \xrightarrow{p} S$ with the body of the inner pspec $Q \xrightarrow{p} T$. The specification $S'(T')$ is obtained by instantiating the target of the outer pspec refinement, $P \xrightarrow{p} S'$, with the inner pspec refinement, $P \Rightarrow T \Rightarrow T'$. Universality of $S(T)$ yields a unique interpretation from $S(T)$ to $S'(T')$, and

$$\begin{tikzcd}
 & P \arrow[dl, "p"'] \arrow[dr, Rightarrow] & & Q \arrow[dl, "p"'] \arrow[dr, Rightarrow] & \\
S \arrow[dr] & & T \arrow[dl, "p"'] \arrow[dr] & & A \arrow[dl, "p"'] \\
 & S(T) \arrow[dr] & & T(A) \arrow[dl, "p"'] & \\
 & & (S(T))(A) = S(T(A)) & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & Q \arrow[dl, "p"'] \arrow[dr, Rightarrow] & \\
S(T) \arrow[dr] & & A \arrow[dl, "p"'] \\
 & (S(T))(A) &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & P \arrow[dl, "p"'] \arrow[dr, Rightarrow] & \\
S \arrow[dr] & & T(A) \arrow[dl, "p"'] \\
 & S(T(A)) &
\end{tikzcd}$$

Figure 13 Associativity of instantiation.

hence a body refinement from the source iterated pspec $Q \xrightarrow{p} S(T)$ to the target iterated pspec $Q \xrightarrow{p} S'(T')$.

If parameter refinements are included, then the parameter refinement of the outer pspec must factor through the body refinement of the inner pspec. In this case, the two target pspecs can be connected to form an iterated pspec and the two pspec interpretations can be combined horizontally, as shown in the bottom half of Figure 14. First, the outer body refinement is transferred to the iterated pspec using the construction in the top half of the figure. Then, a target iterated pspec is constructed by connecting the two target pspecs, $P' \xrightarrow{p} S'$ and $Q' \xrightarrow{p} T'$, with an interpretation $P' \Rightarrow T'$; this interpretation is obtained by factoring the composition $P' \Rightarrow P \Rightarrow T \Rightarrow T'(Q)$ through the morphism $T' \rightarrow T'(Q)$.* The outer target pspec $P' \xrightarrow{p} S'$ is then instantiated to obtain the specification $S'(T')$. Universality of $S'(T')$ yields a unique morphism to $S'(T'(Q))$ and also makes the square from T' to $S'(T'(Q))$ a pushout, hence producing a pspec refinement from the source iterated pspec $Q \xrightarrow{p} S(T)$ to the target iterated pspec $Q' \xrightarrow{p} S'(T')$.

*If this factoring is not possible, then the two pspec interpretations cannot be horizontally composed.

$$\begin{tikzcd}
 & P \arrow[dl, "p"'] \arrow[ddl, "p"] \arrow[drrr, Rightarrow] & & & Q \arrow[dl, "p"'] \arrow[ddl, "p"] \\
S \arrow[d, Rightarrow] \arrow[drr] & & & T \arrow[dl, "p"'] \arrow[d, Rightarrow] & \\
S' \arrow[drr] & & S(T) \arrow[d, Rightarrow, dashed] & T' \arrow[dl, "p"] & \\
 & & S'(T') & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & P \arrow[dl, "p"'] \arrow[ddl, "p"] \arrow[drrr, Rightarrow] & & & & Q \arrow[dl, "p"'] \arrow[ddl, "p"] \\
S \arrow[d, Rightarrow] \arrow[drrr] & & & & T \arrow[dl, "p"'] \arrow[d, Rightarrow] & \\
S'(P) \arrow[drrr] & P' \arrow[uu, Rightarrow] \arrow[dl, "p"'] \arrow[drrr, Rightarrow, dashed] & & S(T) \arrow[d, Rightarrow] & T'(Q) \arrow[dl, "p"'] & Q' \arrow[uu, Rightarrow] \arrow[dl, "p"] \\
S' \arrow[u] \arrow[drrr] & & & S'(T'(Q)) & T' \arrow[u] \arrow[dl, "p"] & \\
 & & & S'(T') \arrow[u] & &
\end{tikzcd}$$

Figure 14 Construction of interpretations for iterated pspecs.

7.1 Commutativity of horizontal composition and instantiation

Given an associative instantiation such as Set(Seq(Nat)) = (Set(Seq))(Nat), we can obtain an interpretation of the resulting specification into two ways: by first refining the inner instantiation and then the outer, or by first composing the two pspec refinements and then refining the instantiation of the iterated pspec. We now show that both paths yield the same interpretation; in other words, associativity of instantiation lifts to pspec interpretations, or horizontal composition of pspec interpretations commutes with instantiation.

In Figure 15, we extend the horizontal composition shown in Figure 14 with instantiations. The top plane shows the associative construction of the instantiation $S(T(A))$ (cf. Figure 13). Similarly, the bottom plane shows the decomposition of $S'(T'(A'))$.

The top right cube shows the construction of the refinement from $T(A)$ to

$T'(A')$ by parallely composing the refinements of $Q \xrightarrow{p} T$ and A (cf. Figure 10). The top left and middle cubes together refine $S(T(A))$ by composing the refinements of $P \xrightarrow{p} S$ and $T(A)$.

Now consider the decomposition $(S(T))(A)$. By omitting the middle and top right cubes, we have a refinement of an iterated pspec (cf. Figure 14). The middle and top right cubes together parallely compose the resulting pspec refinement with the refinement of A.

$$\begin{tikzcd}
 & P \arrow[dl, "p"'] \arrow[ddl, "p"] \arrow[drr, Rightarrow] & & & Q \arrow[dl, "p"'] \arrow[ddl, "p"] \arrow[drr, Rightarrow] & & \\
S \arrow[d, Rightarrow] \arrow[drr] & & & T \arrow[dl, "p"'] \arrow[d, Rightarrow] \arrow[drr] & & & A \arrow[dl, "p"'] \arrow[dd, Rightarrow] \\
S'(P) \arrow[drr] & P' \arrow[uu, Rightarrow] \arrow[dl, "p"'] \arrow[drr, Rightarrow] & S(T) \arrow[d, Rightarrow] \arrow[drr] & T'(Q) \arrow[dl, "p"'] \arrow[ddrr] & Q' \arrow[uu, Rightarrow] \arrow[dl, "p"'] \arrow[drr, Rightarrow] & T(A) \arrow[dl, "p"'] \arrow[dd, Rightarrow] & \\
S' \arrow[u] \arrow[drr] & & S'(T'(Q)) \arrow[ddrr] & T' \arrow[u] \arrow[dl, "p"'] \arrow[drr] & S(T(A)) \arrow[dd, Rightarrow] & & A' \arrow[dl, "p"'] \\
 & & S'(T') \arrow[u] \arrow[drr] & & & T'(A') \arrow[dl, "p"'] & \\
 & & & & S'(T'(A')) & &
\end{tikzcd}$$

Figure 15 Associative instantiation of iterated pspec interpretations.

8 CONCLUDING REMARKS

We have introduced a refinement arrow of mixed variance for parameterized specifications and demonstrated several key properties of this arrow. These properties are evidence that covariant refinement of the body and contravariant refinement of the parameter yields a natural refinement relation for parameter specifications. This naturalness is further reinforced by the commonly used mixed variance subtyping relation between function types. Indeed, parameterized specifications are semantically functions which map models of the parameter to models of the body.

The constructions in this paper are based on relatively weak assumptions about parameterized specifications. The most important assumption is that of being preserved by pushouts. Closure under composition is a kind of transitivity property that seems to be basic. Closure under definitional extensions states that the distinction between theories and presentations is not critical.

However, more investigation may reveal the need for stronger assumptions about parameterized specifications. For example, conservative extensions satisfy the axioms (in logics with an interpolation property), but it is not clear if it is useful to construe them as parameterized specifications.

Finally, we would like to contrast the morphisms and diagrams approach adopted here with the lambda-abstraction approach to parameterization that is used by some authors. Diagram chasing is sometimes more involved than manipulating expressions; consider, for example, refinement of iterated pspecs. However, it appears that the flexibility given by explicit morphisms is almost always required in practice. Real software is incredibly detailed and demands powerful bookkeeping tools.

8.1 Related Work

A refinement arrow of mixed variance was described by Ehrig & Mahr (1990) and by Sannella & Wirsing (1982). Using the notation of this paper, their refinement arrow is a pair of interpretations such that the square in the left half of Figure 16 commutes.. Such a square can be decomposed into a body refinement followed by a parameter refinement, as shown in the right half of Figure 16. Thus, this refinement arrow is somewhat less general than ours: in ours, the body interpretation need not factor through the target instantiation. We believe that, in most cases, it is not possible to find an interpretation directly between the source and target bodies. Indeed, the purpose of connecting the two parameters and instantiating the target pspec is to provide a sufficiently strong base for refining the body of the source pspec (cf. Figure 8).

$$\begin{tikzcd}
S \arrow[d, Rightarrow] & P \arrow[l, "p"] \\
S' & P' \arrow[l, "p"] \arrow[u, Rightarrow]
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[dd, Rightarrow] \arrow[dr, Rightarrow] & & P \arrow[ll, "p"] \arrow[dl, "p"] \\
& S'(P) & \\
S' \arrow[ur] & & P' \arrow[ll, "p"] \arrow[uu, Rightarrow]
\end{tikzcd}$$

Figure 16 Another refinement arrow for pspecs.

Ehrig & Große-Rhode (1994) adopt an axiomatic approach to parameterized specifications to abstract away from the various semantics: a pspec denotes a class of functors closed under composition and amalgamated extension. This is similar to our axioms about pspecs, provided they are transferred to the semantic level via a model functor *Mod* which transforms pushouts to pullbacks. They consider covariant refinement in this framework. Covariant refinement can be treated as instantiation followed by body refinement, and its properties follows from this decomposition. On the other hand, mixed-variance refinement seems to be fundamental, in the sense that it cannot be built up from other primitive notions.

Acknowledgements

Discussions with Richard Jüllig clarified and simplified some of the concepts in this paper; in particular, the use of interpretations to instantiate parameterized specifications was proposed by him. This work was supported by the Office of Naval Research contract N00014-93-C-0274.

9 APPENDIX: PUSHOUT OF INTERPRETATIONS

LEMMA 9.1: *Epimorphism pushout.* In any category, if an arrow factors via an epimorphism as in the diagram on the left below, the factoring is also a pushout, as in the diagram on the right below.

$$\text{if} \quad \begin{tikzcd} A \arrow[r, two heads, "g"] \arrow[dr, "f"'] & C \arrow[d, "h"] \\ & B \end{tikzcd} \quad \text{then} \quad \begin{tikzcd} A \arrow[r, two heads, "g"] \arrow[d, "f"'] & C \arrow[d, "h"] \\ B \arrow[r, "\mathrm{id}"'] \arrow[ur, phantom, very near end, "\ulcorner"] & B \end{tikzcd}$$

PROOF. Given any cocone $B \xrightarrow{j} E \xleftarrow{k} C$, we must find a unique arrow $B \xrightarrow{w} E$ such that $j = \mathrm{id}_B; w$ and $k = h; w$ (see diagram below). The former equation uniquely determines w to be equal to j. The latter equation follows from the following calculation:

$$\begin{array}{lll} & f; j = g; k & \textit{—cocone into } E \\ \iff & (g; h); j = g; k & \textit{—hypothesis about } f \\ \iff & h; j = k & \textit{—since } g \textit{ is epi} \\ \iff & h; w = k & \textit{—since } j = w \end{array}$$

$$\begin{tikzcd} A \arrow[r, two heads, "g"] \arrow[d, "f"'] & C \arrow[d, "h"] \arrow[ddr, bend left, "k"] & \\ B \arrow[r, "\mathrm{id}"'] \arrow[drr, bend right, "j"'] & B \arrow[dr, dashed, "w"'] & \\ & & E \end{tikzcd}$$

□

COROLLARY 9.2: *Factoring a definitional extension.*

$$\text{if}\quad \begin{tikzcd} A \arrow[r, "d"] \arrow[dr, "d"'] & C \arrow[d] \\ & B \end{tikzcd} \qquad \text{then}\quad \begin{tikzcd} A \arrow[r, "d"] \arrow[dr, "d"'] & C \arrow[d, "d"] \\ & B \end{tikzcd}$$

PROOF. A definitional extension is an epimorphism, so the triangle on the left is a pushout, and pushouts preserve definitional extensions. □

COROLLARY 9.3: *Extra definitions in interpretation mediator.*

$$\text{if}\quad \begin{tikzcd} & B_+^1 \arrow[dd, "d"] & \\ A \arrow[ur] \arrow[dr] & & B \arrow[ul, "d"'] \arrow[dl, "d"] \\ & B_+^2 & \end{tikzcd} \qquad \text{then}\quad \begin{tikzcd} A \arrow[r, Rightarrow, shift left=2] \arrow[r, Rightarrow, shift right=2] \arrow[r, phantom, "="] & B \end{tikzcd}$$

□

COROLLARY 9.4: *Factoring via a definitional extension.* The commuting diagram of morphisms on the left below may be construed as the commuting diagram of interpretations on the right below.

$$\text{if}\quad \begin{tikzcd} A \arrow[r, "d"] \arrow[dr] & C \arrow[d] \\ & B \end{tikzcd} \qquad \text{then}\quad \begin{tikzcd} A \arrow[dr] & C \arrow[l, Rightarrow] \arrow[d] \\ & B \end{tikzcd}$$

□

LEMMA 9.5: *Universality of interpretation pushout.* The pushout of two interpretations, constructed as shown in Figure 4, is universal.
PROOF.
Existence of witness. In the diagram on the left below, $B \overset{f}{\Longleftarrow} A \overset{g}{\Longrightarrow} C$ are the two given interpretations, and the morphisms $B \xrightarrow{g'} D \xleftarrow{f'} C$ form the pushout cocone (in the category of interpretations). Given any other cocone $B \overset{j}{\Longrightarrow} A \overset{k}{\Longleftarrow} C$, we must find an interpretation $w\colon D \Rightarrow E$ which factors this cocone through the pushout cocone.

In the diagram on the right below, the pushout and the given cocone are expanded out to show the effect of the equation $f; j = g; k$. From the definitions of composition and equality for interpretations, we get the (pushout) specifications E_3, E_4 and E_5 and the equation $(A \to B_+ \to E_3 \to E_5) =$

($A \to C_+ \to E_4 \to E_5$). Universality of D (in the category of morphisms) yields the witness morphism $D \to E_5$.

We thus get the witness interpretation $w\colon (D \to E_5 \leftarrow E)$ (in the category of interpretations). It is a straightforward diagram chase and application of Corollary 9.3 to check that this witness satisfies the required properties of $g'; w = j$ and $f'; w = k$.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[rr, Rightarrow, bend left, "g"] \arrow[dd, Rightarrow, bend right, "f"'] & C_+ \arrow[d] & C \arrow[l, "d"'] \arrow[dl, "f'"] \arrow[dddr, Rightarrow, "k"] & \\
B_+ \arrow[r] & D \arrow[ddrr, Rightarrow, dashed, "w"] & & \\
B \arrow[u, "d"] \arrow[ur, "g'"'] \arrow[drrr, Rightarrow, "j"'] & & & \\
& & & E
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r] \arrow[d] & C_+ \arrow[d] \arrow[ddrr] & C \arrow[l, "d"'] \arrow[dl] \arrow[ddrr] & & \\
B_+ \arrow[r] \arrow[ddrr] & D \arrow[dr, dashed] & & & \\
B \arrow[u, "d"] \arrow[ur] \arrow[ddrr] & & E_5 & E_4 \arrow[l, "d"'] & E_2 \arrow[l, "d"'] \\
& & E_3 \arrow[u, "d"] & & \\
& & E_1 \arrow[u, "d"] & & E \arrow[ll, "d"] \arrow[ull, "d"] \arrow[uull, "d"] \arrow[uul, "d"] \arrow[uu, "d"']
\end{tikzcd}$$

Uniqueness of witness. Suppose there are two interpretations $v, w\colon D \Rightarrow E$ which factor the cocone $\langle j, k\rangle$ through the pushout cocone $\langle g', f'\rangle$, as shown in the diagram below. We will show that these two interpretations are equal, thus making the witness unique.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[rr, Rightarrow, bend left, "f"] \arrow[dd, Rightarrow, bend right, "g"'] & C_+ \arrow[d] & C \arrow[l, "d"'] \arrow[dl, "f'"] \arrow[dddr, Rightarrow, "k"] & \\
B_+ \arrow[r] & D \arrow[ddrr, Rightarrow, shift left, "w"] \arrow[ddrr, Rightarrow, shift right, "v"'] \arrow[ddrr, phantom, "?"] & & \\
B \arrow[u, "d"] \arrow[ur, "g'"'] \arrow[drrr, Rightarrow, "j"'] & & & \\
& & & E
\end{tikzcd}$$

Using Corollary 9.4, we can construct from the given cocone $\langle j, k\rangle$ another cocone $\langle j', k'\rangle$ from the morphism pair $B_+ \leftarrow A \to C_+$ to E, such that the two interpretations v and w factor this cocone through the pushout cocone $B_+ \to D \leftarrow C_+$. This construction is shown in the diagram on the left below.

In the diagram on the right below, the interpretations are expanded and forced to have a common mediator, E_8. Universality of the pushout D (in the category of morphisms) implies that the two morphisms $D \to E_6 \to E_8$ and $D \to E_7 \to E_8$ are equal, because they both factor the cocone $\langle (B_+ \to E_3 \to E_8), (C_+ \to E_4 \to E_8) \rangle$. Thus the two interpretations $v\colon (D \to E_6 \leftarrow E)$ and $w\colon (D \to E_7 \leftarrow E)$ are equal.

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & C_+ \arrow[r, dotted] \arrow[d] \arrow[dddrr, Rightarrow, "k'"] & C \arrow[dl, dotted] \arrow[dddr, dotted] & \\
B_+ \arrow[r] \arrow[d, dotted] \arrow[ddrrr, Rightarrow, "j'"'] \arrow[ur, phantom, "\lrcorner", very near end] & D \arrow[ddrr, Rightarrow, shift left, "w"] \arrow[ddrr, Rightarrow, shift right, "v"'] \arrow[ddrr, phantom, "?"] & & \\
B \arrow[ur, dotted] \arrow[drrr, dotted] & & & \\
 & & & E
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r] \arrow[d] & C_+ \arrow[d] \arrow[drrr] & & & \\
B_+ \arrow[r] \arrow[dddrr] \arrow[ur, phantom, "\lrcorner", very near end] & D \arrow[drr] \arrow[ddr] & & & \\
 & & ? & E_7 \arrow[d, "d"] & E_4 \arrow[dl, "d"'] \\
 & & E_6 \arrow[r, "d"] & E_8 & \\
 & & E_3 \arrow[ur, "d"] & & E \arrow[ll, "d"] \arrow[ul, "d"'] \arrow[ull, "d"] \arrow[uul, "d"] \arrow[uu, "d"']
\end{tikzcd}$$

□

REFERENCES

Bidoit, M. (1987), The stratified loose approach: A generalization of initial and loose semantics, *in* ‘Recent Trends in Data Type Specification’, Vol. 332 of *Lecture Notes in Computer Science*, Springer-Verlag, Gullane, Scotland, pp. 1–22.

Burstall, R. M. & Goguen, J. A. (1977), Putting theories together to make specifications, *in* ‘Proc. 5th Int. Joint Conf. on Artificial Intelligence’, pp. 1045–1058.

Ehrig, H. & Große-Rhode, M. (1994), ‘Functorial theory of parameterized specifications in a general specification framework’, *Theoretical Computer Science* **135**, 221–266.

Ehrig, H. & Kreowski, H. J. (1982), Parameter passing commutes with implementation of parameterized data types, *in* ‘9^{th}ICALP’, Vol. 140 of *Lecture Notes in Computer Science*, Springer-Verlag, Aarhus, Denmark, pp. 197–211.

Ehrig, H., Kreowski, H. J., Thatcher, J., Wagner, E. & Wright, J. (1981), Parameter passing in algebraic specification languages, *in* ‘Proceedings, Workshop on Program Specification’, Vol. 134 of *Lecture Notes in Computer Science*, Springer-Verlag, Aarhus, Denmark, pp. 322–369.

Ehrig, H. & Mahr, B. (1990), *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*, Vol. 21 of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin.

Ganzinger, H. (1983), 'Parameterized specifications: Parameter passing and implementation with respect to observability', *ACM Transactions on Programming Languages and Systems* **5**(3), 318–354.

Goguen, J. A. & Meseguer, J. (1982), Universal realization, persistent interconnection and implementation of abstract modules, *in* '9^{th}ICALP', Vol. 140 of *Lecture Notes in Computer Science*, Springer-Verlag, Aarhus, Denmark, pp. 265–281.

Lambek, J. & Scott, P. J. (1986), *Introduction to Higher Order Categorical Logic*, Cambridge University Press, Cambridge.

Sannella, D. & Tarlecki, A. (1988), 'Towards formal development of programs from algebraic specifications: Implementations revisited', *Acta Informatica* **25**, 233–281.

Sannella, D. & Wirsing, M. (1982), Implementation of parameterized specifications, *in* '9^{th}ICALP', Vol. 140 of *Lecture Notes in Computer Science*, Springer-Verlag, Aarhus, Denmark, pp. 473–488. Extended abstract.

Srinivas, Y. V. & Jüllig, R. (1995), Specware:$^{\text{tm}}$ formal support for composing software, *in* B. Moeller, ed., 'Proceedings of the Conference on Mathematics of Program Construction', Springer-Verlag, Berlin, pp. 399–422. Lecture Notes in Computer Science, Vol. 947.

Wirsing, M. (1986), 'Structured algebraic specifications: A kernel language', *Theoretical Computer Science* **42**, 123–249.