Category localization semantics for specification refinements

Jerzy Tomasik* Jerzy Weyman †

Abstract: *We describe the theory of refinements of specifications based on localizations of categories. We show that the class of definitional extensions in the category of specifications satisfies the axioms of localizations of categories. This allows to enrich in a consistent manner the family of specification morphisms and view certain specification refinements as arrows in the localized category. Such description provides a useful guide for finding a path from a given specification to a specification which is a provably correct code in a programming language (C++, Lisp, Java).*
Keywords: Category localization, specification refinements, Specware, fusion.

1 Introduction

Building complex logical systems for software specifications led Goguen, Burstal, Tarlecki et al.(cf. [14], [8], [11]), to propose a program of the unification of specification fusion methods in order to provide a framework for systematic construction of new, more and more complex logical systems via limit and/or colimit operations. In this note we go one step further along this line. We provide a category theory semantics for the refinement operator linking high level specification with its provably correct code in a programming language (e.g. $C++, Lisp, Java$).The localization semantics of step-by-step refinements is described in the case of the sensory fusion only. It can be easily extended to more general cases.

The basic idea of category theory applications to software engineering and sensory fusion theory was to represent parts of a program by specifications and then use the operation of limit or colimit to merge them in a consistent way into a bigger program. Similarly, the specifications corresponding to sensors would be glued by the colimit operation to the specification describing the fused system of sensors. In fact to obtain a (provably correct) code the operation of fusion consists of two equally important steps:

First step needs to obtain from specifications of sensors a specification describing a fused system of sensors, basically using colimits.

The second step consists of refining this specification into a code in a programming language which allows to analyze fused information.

Jullig and Srinivas [9] defined refinements in a category theory setting. They define the refinement arrow $X \to Z$ as a diagram

$$X \xrightarrow{f} Y \xleftarrow{s} Z \qquad (*)$$

where f is a morphism of specifications and s is a definitional extension. They define also the composition of two refinement arrows.

The definition in [9] is not very precise. In fact they do not address the question whether the refinement arrow is an arrow in a category and in which one. This is an essential difficulty because in practice one needs to do multi-step refinements and at each stage one should know how to proceed.

In this note we show that indeed the refinement arrow is an arrow in a localized category. More precisely, the refinement arrow becomes an arrow in the category of specifications localized at the system of morphisms called definitional extensions (we use here the usual definition of the category *Spec* of specifications like in [8] or [9]). Such localized categories can be constructed for very general classes of morphisms (cf. [3],[6],[5]), so this is not in itself very useful. It turns out however that the usual axioms for (left) localization are satisfied in this case (comp. [6]). This is of fundamental importance since it allows to describe objects and morphisms in the localized category explicitly.

*LLAIC1, Université Clermont 1 ,BP 86, 63172 Aubière cedex (France). Mail: tomasik@llaic.u-clermont1.fr
†Northeastern University, Boston MA, (USA) . Mail: j.weyman@neu.edu

Section 2 contains basic notions of category localizations. We follow the approach of Gabriel and Zisman [6]. In particular we define the classes of morphisms in arbitrary categories allowing a calculus of left fractions. For each such class of morphisms we can describe the objects and morphisms in a localized category explicitly. In the section 3 we show that the class of definitional extensions in the category of specifications allows the calculus of left fractions. Moreover, the definitional extensions have one extra property which allows to improve the description of morphisms.

2 Preliminaries

The category theory was applied to systems theory and systems engineering [2], [7]. This theory was embodied in the software development tool Specware(tm) [12], [13], [16], [15], [18], [19].

2.1 Category of Signatures

A signature consists of the following:

1. A set S of sort symbols
2. A triple $O =< C, F, P >$ of operators

Where C is a set of sorted constant symbols, F is a set of sorted function symbols, and P is a set of sorted predicate symbols.

A signature morphism is a consistent mapping from one signature to another (from sort symbols to sort symbols, and from operator symbols to operator symbols).

The category SIG consists of objects that are signatures and morphisms that are signature morphisms. Composition of two morphisms is the composition of the two mappings.

2.2 Category of Specifications

A specification consists of:

1. A signature $Sig =< S, O >$,
2. A set Ax of axioms over Sig

Given two specifications $< Sig1, Ax1 >$ and $< Sig2, Ax2 >$, a signature morphism M between $Sig1$ and $Sig2$ is a *specification morphism* between the specifications iff: $\forall a \in Ax1, (Ax2 \vdash M(a))$. That is, a specification morphism between specification A and specification B is a mapping of the signature of A into the signature of B such that all the axioms of A (translated under the signature mapping) are theorems in the theory of B. In other words, it maps the signature of A into B such that all the properties of A are preserved in B (of course, additional constraints may be added in B). Specification B is a *refinement* of specification A.

2.3 Diagrams and Colimits

A diagram in a category C is a collection of vertices and directed edges consistently labelled with objects and arrows (morphisms) of C. Let D be a diagram in C.

A *cocone* for D is

1. A C-object x (called the apex),
2. For each D-object d_i, a C-arrow f_i (called a *leg*) from d_i to x, such that for each D-arrow g from d_i to d_j, $f_j \circ g = f_i$.

A *colimit* for diagram D is a cocone with apex x such that for any other cocone with apex y there is a unique C-arrow $f_{x,y}$ such that for any D-object d, composing the colimit leg with $f_{x,y}$ yields the leg from d to y.

In the category of specifications and specification morphisms, the apex of a colimit of a diagram intuitively corresponds to the most general specification that "welds together" the specifications in the diagram (welding them together in ways that the specification morphisms dictate).

2.4 Specware - a tool for Software Development

Specware(tm) [13], [16] is a software development environment supporting the specification, design, and semi-automated synthesis of correct-by-construction software. It represents the confluence of capabilities of earlier prototype systems (KIDS [12], REACTO [17], DTRE [12]), grounded on a strong mathematical foundation (SLANG, based on category theory). The current version of Specware(tm) is a robust implementation of this foundation. Basic notions in Specware(tm) are that from the category *Spec* of specifications of multi-sorted algebras, i.e. specification, specification morphism, colimit of a diagram, which are used for building larger specifications from smaller ones etc. Specware(tm) supports automation of:

1. component-based specification of programs using a graphical interface,
2. incremental refinement of specifications into correct code in various target languages (e.g., currently C++ and LISP, and potentially Ada and COBOL),
3. recording and experimenting with different design decisions,
4. domain-knowledge capture, verification and manipulation
5. design and synthesis of software architectures/frameworks
6. design and synthesis of algorithm schemas
7. design and synthesis of reactive systems
8. data-type refinement
9. program optimization

Example 2.1 - ***Pre-order as a colimit of specifications.***
One specification is that of a binary relation (Bin), another is that of a reflexive binary relation (Ref), and a third that of a transitive binary relation (Trans). There is a specification morphism between the binary relation specification and each of the other specifications. This diagram is depicted below.

$$Ref \overset{rr}{\leftarrow} Bin \overset{tt}{\rightarrow} Trans$$

Colimit of a diagram is a specification that is the most general specification that is a refinement of each of the specifications in the diagram.

$$\begin{array}{ccc} Bin & \overset{tt}{\rightarrow} & Trans \\ rr \downarrow & & a \downarrow \\ Ref & \overset{b}{\rightarrow} & Pre-Ord \end{array}$$

In our example, colimit of the diagram is a specification of a binary relation that is both reflexive and transitive; in other words, a pre-order relation($Pre-Ord$).

2.5 Synthesize Software by interpretations

To synthesize executable software for a specification, it is necessary for the sorts and operations of that specification to be mapped to data structures and executable functions in a target programming language (e.g. *Lisp*). For example [15], [18], to generate software for an abstract theory of sets we will require a function for inserting a new element in the set. Since set is not a primitive data structure in Lisp, there is no pre-existing insert function that the abstract function can be mapped to in the target programming language. However, we can use the list data structure in *Lisp* to represent sets, and use existing list operations to define an algorithm for inserting a member into a set. We must ensure that the list obeys the set axioms; e.g., prohibiting duplicate members. The algorithm which does this is:

begin
if the new element is already in the list,

then return the list
else use the list insert operation to add the element to the list;
return (the new list)
end.

To do this we have to define an *interpretation* from set to list. In general, interpretations describe how specifications can be implemented in terms of simpler or more primitive specifications. To create an interpretation from specification(set) to spec(list), one must do the following:

1. Create a third specification, called a *mediator*, which is a *definitional extension* of specification list which includes new sorts and operations, and their definitions, for manipulating sets as lists.

2. Define a specification morphism from set to the mediator specification.

In step one - the new definitions are created so that the morphism in step two can be defined. An interpretation may be created from one specification to another, without necessarily generating code, as from set to list.

The main problem appears in step two. - Is there any morphism from set to the mediator specification in the category *Spec*?

We answer the question in the next sections and we prove that such a morphism exists in the category *Spec* localized under the family of definitional extensions isomorphisms.

3 Localizations of categories

Let $\mathcal{C}$ be a category. Let Σ be a class of morphisms in $\mathcal{C}$.

Definition 3.1 *The localization category $\mathcal{C}[\Sigma^{-1}]$ of the category $\mathcal{C}$ at the class Σ is the pair $(\mathcal{C}[\Sigma^{-1}], P_\Sigma)$ where $\mathcal{C}[\Sigma^{-1}]$ is a category and $P_\Sigma : \mathcal{C} \to \mathcal{C}[\Sigma^{-1}]$ is a functor with the following universal property.*

1. *For every $s \in \Sigma$ the morphism $P_\Sigma(s)$ is an isomorphism,*

2. *For every functor $F : \mathcal{C} \to \mathcal{D}$ such that for every $s \in \Sigma$ the morphism $F(s)$ is an isomorphism in $\mathcal{D}$, there exists a unique functor $G : \mathcal{C}[\Sigma^{-1}] \to \mathcal{D}$ such that $F = G \circ P_\Sigma$.*

The existence of localizations is proved in [3] and in [6]. The localization of a category has a very concrete description under some mild assumptions on Σ.

Definition 3.2 *The class of morphisms Σ in a category $\mathcal{C}$ admits left fractions if the following conditions are satisfied.*

1. *All identity morphisms are in Σ,*

2. *The composition $t \circ s$ of morphisms $s : X \to Y$, $t : Y \to Z$ from Σ is also in Σ.*

3. *Every diagram*

$$X' \xleftarrow{s} X \xrightarrow{f} Y$$

with $s \in \Sigma$ can be extended to a commutative square

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ s \downarrow & & t \downarrow \\ X' & \xrightarrow{f'} & Y' \end{array}$$

with $t \in \Sigma$.

4. *Let the morphisms $f, g : X \to Y$ and a morphism $s : X' \to X$ from Σ satisfy $f \circ s = g \circ s$. Then there exists a morphism $t : Y \to Y'$ from Σ such that $t \circ f = t \circ g$.*

Remark 3.3 *The way to understand the above conditions is to think of the diagram*

$$\begin{array}{ccc} d & & c \\ & f \searrow \quad \swarrow s & \\ & c' & \end{array}$$

as of a fraction $s^{-1} \circ f$. *Then the condition* (1.) *says that the morphism the fraction* $id^{-1} \circ f$ *can be identified with* f, *condition* (2.) *says that the product of denominators is a denominator, and condition* (3.) *says that the right fraction* $f \circ s^{-1}$ *can be rewritten as a left fraction* $t^{-1} \circ f'$. *Conditions* (2.) *and* (3.) *assure that product of left fractions can be written again as a left fraction.*

Let c be an arbitrary object of the category $\mathcal{C}$. We define the category $c \setminus \Sigma$ as a subcategory of the category $c \setminus \mathcal{C}$ generated by morphisms $s : c \to c'$ from Σ.

Every object $d \in \mathcal{C}$ defines a functor from $c \setminus \Sigma$ to Set associating to any object $s : c \to c'$ from $c \setminus \mathcal{C}$ the set $Hom_{\mathcal{C}}(d, c')$. Let $lim_s Hom_{\mathcal{C}}(d, \tau s)$ be the limit of this functor.

Explicitly, for $s \in \Sigma$, this limit is given as the set of equivalence classes of the relation on the set $H(d, c)$ of diagrams

$$\begin{array}{ccc} d & & c \\ & f \searrow \quad \swarrow s & \\ & c' & \end{array}$$

This is the smallest relation making the pairs

$$\begin{array}{ccc} d & & c \\ & f \searrow \quad \swarrow s & \\ & c' & \end{array}$$

$$\begin{array}{ccc} d & & c \\ & g \searrow \quad \swarrow t & \\ & c'' & \end{array}$$

equivalent whenever there exists a morphism $\gamma : c' \to c''$ in $\mathcal{C}$ such that $t = \gamma \circ s$, $g = \gamma \circ f$. It follows from the conditions $(1.) - (4)$ of Definition3.2 that two pairs

$$\begin{array}{ccc} d & & c \\ & f \searrow \quad \swarrow s & \\ & c' & \end{array}$$

$$\begin{array}{ccc} d & & c \\ & g \searrow \quad \swarrow t & \\ & c'' & \end{array}$$

are equivalent if and only if there exists a commutative diagram

$$\begin{array}{ccccc} & & c'' & & \\ & g \nearrow & t \uparrow & \searrow b & \\ d & & c & & c''' \\ & f \searrow & s \downarrow & \nearrow a & \\ & & c' & & \end{array}$$

for which the morphisms $as = bt$ belong to Σ. This means that the elements of $lim_s Hom_{\mathcal{C}}(d, \tau s)$ can be thought of as left fractions $s^{-1} \circ f$.

Define a category $\Sigma^{-1}\mathcal{C}$ as follows. The objects of $\Sigma^{-1}\mathcal{C}$ are the objects of $\mathcal{C}$. We set

$$Hom_{\Sigma^{-1}\mathcal{C}}(c, d) = lim_s Hom_{\mathcal{C}}(c, \tau s), s \in c \setminus \Sigma$$

The composition of fractions $s^{-1} \circ f \in Hom_{\Sigma^{-1}\mathcal{C}}(d, c)$ and $t^{-1} \circ g \in Hom_{\Sigma^{-1}\mathcal{C}}(e, d)$ is defined as $(s's)^{-1} \circ (f'g)$ where $s' \in \Sigma$, f' are the morphisms closing the commutative diagram

$$\begin{array}{ccccc} e & & d & & c \\ & g\searrow & t\downarrow & f\searrow & \downarrow s \\ & & d' & & c' \\ & & & f'\searrow & \downarrow s' \\ & & & & c'' \end{array}$$

Now we compare the category $\Sigma^{-1}\mathcal{C}$ with $\mathcal{C}[\Sigma^{-1}]$. Let $d, c \in Ob(\mathcal{C})$. The map

$$(s, f) \mapsto (P_\Sigma s)^{-1} P_\Sigma(f)$$

of the set $H(d, c)$ into $Hom_{\mathcal{C}[\Sigma^{-1}]}(d, c)$ maps the equivalent elements to the same element. Therefore the correspondence

$$s^{-1} \circ f \mapsto (P_\Sigma s)^{-1} P_\Sigma(f)$$

defines a function

$$\pi(d, c) : Hom_{\Sigma^{-1}\mathcal{C}}(d, c) \to Hom_{\mathcal{C}[\Sigma^{-1}]}(d, c).$$

This gives rise to a functor

$$\pi : \Sigma^{-1}\mathcal{C} \to \mathcal{C}[\Sigma^{-1}]$$

Theorem 3.4 *(Gabriel-Zisman [6]) Let $\mathcal{C}$ be a category and let Σ be a class of morphisms admitting left fractions. Then the functor π is an equivalence of categories.*

We can prove the result which is very useful in the framework of information fusion applications, usually realized by means of the colimit operation (see e.g.[8], [9]).

Theorem 3.5 *Let $\mathcal{C}$ be a category and let Σ be a class of morphisms admitting left fractions. Then the functor $P_\Sigma : \mathcal{C} \to \mathcal{C}[\Sigma^{-1}]$ commutes with finite colimits.*

Whence if the fusion operator is the colimit of a finite number of specifications the multi step refinement and fusion operations can be realized in an arbitrary order.

The ideas of representing a diagram

$$X \leftarrow X' \to Y$$

as a morphism in some category appeared since in different contexts. For example Ehrig (cf.[4]) defines for a category $\mathcal{C}$ with pushouts the category $Cospan(\mathcal{C})$ as follows.

Example 3.6 *The objects in $Cospan(\mathcal{C})$ are the objects of $\mathcal{C}$ and the morphisms from X to Y are the diagrams*

$$X \xrightarrow{f} Y' \xleftarrow{s} Y.$$

The composition of this morphism and the morphism

$$Y \xrightarrow{g} Z' \xleftarrow{t} Z$$

is the morphism

$$X \xrightarrow{g'f} Y'' \xleftarrow{s't} Z$$

where

$$\begin{array}{ccc} Y & \xrightarrow{g} & Z' \\ s\downarrow & & s'\downarrow \\ Y' & \xrightarrow{g'} & Y'' \end{array}$$

is a pushout diagram.

The pushout requirement is the analogue of the axiom (3.) of the set of left fractions in the Definition3.2. Let Σ be the class of all morphisms in $\mathcal{C}$. We get a natural functor $Cospan(\mathcal{C}) \to \Sigma^{-1}\mathcal{C}$ relating $Cospan(\mathcal{C})$ to $\Sigma^{-1}(\mathcal{C})$.

4 Definitional extensions

Let *Spec* be the category of specifications. Consider the class Λ of definitional extensions in *Spec*. The observation that allows to apply the localizations to our purposes is

Theorem 4.1 *The class Λ of morphisms in Spec admits left fractions.*

Proof:

We need to verify the four condition from the Definition 3.2. The conditions (1.) and (2.) are obvious. To check the condition (3.) consider the diagram

$$X' \stackrel{s}{\leftarrow} X \stackrel{f}{\rightarrow} Y$$

where X, Y, X' are specifications, f, s are morphisms of specifications and s is a definitional extension. We define a specification Y' as follows. The sorts of Y' are the sorts of Y and the sorts of the type $f(x)$ where x is a sort in X. Similarly for operations in Y'. The axioms in Y' are the axioms in Y and the images by f of all definitions of sorts, operations, etc. from X' in terms of sort, operations, etc. from X. Let $f' : X' \rightarrow Y'$, $t : Y \rightarrow Y'$ be obvious morphisms of specifications. Then $t \circ f = f' \circ s$. Moreover, t is a definitional extension, because every object of type $f(x)$ is defined in term of objects of type $t(y)$.

To prove condition (4.) we observe that if $f, g : X \rightarrow Y$, $s : X' \rightarrow X$ are morphisms of specifications, and if s is a definitional extension, then if $f \circ s = g \circ s$ then $f = g$. Therefore it is enough to take $t = id : Y \rightarrow Y$. □

Remark 4.2 *In the course of the proof we showed also the following fact:*

Let Spec be a category of specifications and Λ - a class of definitional extensions. Then Λ satisfies the condition (4.) *of Definition 3.2 in a stronger form: we can choose the objects X' and Y' and morphisms f' and t so that t is an identity morphism.*

5 Conclusion.

Refinements in the language of Jullig-Srinivas [9] have a natural interpretation as morphisms in the localized category $Ref = Spec[\Lambda^{-1}]$ called *the refinement category*. Notice that the category Ref has the same objects as *Spec* and the morphisms can be explicitly described as in [6].

This allows to put the theory of refinements on a solid footing. Moreover one can see that in the case of $\mathcal{C} = Spec$, $\Sigma = \Lambda$ the axiom (4.) from the Definition 3.2 is true in the stronger form - it is enough to take $t = id$ to fulfill the axiom. Thus it should lead to even simpler description of the set of morphisms in Ref than the general description in [6].

The ability to recognize identical specifications is a guide in finding the shortest path of refinements from initial rough specification to a code in a programming language. Our construction can be considered as a general method of narrowing choices in finding of this path for the colimit based and iterated fusion procedure

Acknowledgements. The authors are grateful to Mieczyslaw M.Kokar for drawing their attention to the sensory fusion problem and for his helpful suggestions.

6 Bibliography.

References

[1] L. Blaine, A.Goldberg. *DTRE - A Semi-Automatic Transformation System, in Constructing Programs from Specifications* in Constructing Programs from Specifications, ed. B. Moller, North Holland, 1991.

[2] R.M. Burstall, J. A.Goguen. *The Semantics of Clear, a Specification Language* Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification, LNCS, 86, Springer-Verlag,292-332.

[3] P.M. Cohn. *Free Rings and their Relations* Academic Press, 1985

[4] H. Ehrig. *Bigraphs meet Double Pushouts* EATCS Bulletin, 78, 72-85, October, 2002.

[5] M. Faber, P. Vogel. *The Cohn Localization of the Free Group Rings.* Math.Proc.Comb.Phil.Soc., III 433, 1992.

[6] P. Gabriel, M. Zisman. *Calculus of Fractions and Homotopy Theory.* Springer-Verlag, Berlin, Heidelberg, New York,1967. [Goguen, J. A., 1971] Mathematical Representation of Hierarchically Organized Systems, in Global Systems Dynamics, ed. E. Attinger and S. Karger, pages 112-128.

[7] J.A. Goguen. *Mathematical Representation of Hierarchically Organized Systems.* In Global Systems Dynamics, ed. E. Attinger and S. Karger, 112-128.

[8] J.A. Goguen, R.M. Burstall. *A study in the foundations of programming methodology: Specifications, institutions, charters and parchments.* In D.Pitt et al., ed.,*Category Theory and Computer Programming*, LNCS **240** 313-333.

[9] Jullig, Srinivas, *Specware Manual.* Kestrel Institute, 1992.

[10] M. Kokar, J. Tomasik, J. Weyman. *Application of localizations of categories to fusion. Preprint* No.90 LLAIC,2000.

[11] T. Mossakowski, A. Tarlecki, W. Pawłowski. *Combining and representing logical systems.* In E.Moggi, ed.,*Category Theory and Computer Programming*, LNCS **1290** 177-198,Springer 1997.

[12] D.Smith, *KIDS: A Knowledge Based Software Development System, in Automating Software Design.* In Automating Software Design, Eds. M. Lowry and R. McCartney, MIT Press, 1991.

[13] Y.V.Srinivas, R.Jullig, *Specware(tm): Formal Support for Composing Software.* In Proceedings of the Conference of Mathematics of Program Construction, Kloster Irsee, Germany, 1995.

[14] A. Tarlecki, J.A. Goguen, R.M. Burstall. *Some fundamental algebraic tools for the semantics of computation. PartIII: Indexed categories.* Theoretical Computer Science **91** 239-264, 1991.

[15] M.Uschold et al., *Ontology Reuse and Application* in Proceedings of the First International Conference on Formal Ontology in Information Systems, Trento, 1998.

[16] R.Waldiger et al., *Specware Language Manual 2.0.1* Suresoft,Inc., 1996.

[17] T.C.Wang, A.Goldberg, *A Mechanical Verifier for Supporting the Design of Reliable Reactive Systems* International Symposium on Software Reliability Engineering, Austin, Texas, 1996.

[18] K.Wiliamson, M.Healy, *Boeing Applied Research Technology* Bellevue, Washington March 20th, 1998.

[19] K.Wiliamson, P.Ridle, *Knowledge Repositories for Multiple Uses, Goddard Conference on Space Applications of Artificial Intelligence* Goddard Conference on Space Applications of Artificial Intelligence, *Deriving Engineering Software from Requirement*, 353-367, 1991.