Modeling Inheritance as Coercion in a Symbolic Computation System*

César Domínguez
Departamento de Matemáticas y Computación
Universidad de La Rioja
Edificio Vives, Luis de Ulloa s/n
E-26004 Logroño (La Rioja, Spain)
cedomin@dmc.unirioja.es

Julio Rubio
Departamento de Matemáticas y Computación
Universidad de La Rioja
Edificio Vives, Luis de Ulloa s/n
E-26004 Logroño (La Rioja, Spain)
jurubio@dmc.unirioja.es

ABSTRACT

In this paper the analysis of the data structures used in a symbolic computation system, called Kenzo, is undertaken. We deal with the specification of the inheritance relationship since Kenzo is an object-oriented system, written in CLOS, the Common Lisp Object System. We focus on a particular case, namely the relationship between simplicial sets and chain complexes, showing how the order-sorted algebraic specifications formalisms can be adapted, through the "inheritance as coercion" metaphor, in order to model this Kenzo fragment.

1. INTRODUCTION

Kenzo is a Sergeraert's system [28] designed for the calculation of homology and homotopy groups for topological spaces. It has been written in CLOS and is a descendant of the first Sergeraert's system for symbolic computation in Algebraic Topology, called EAT [27].

The main difference between Kenzo and EAT, apart from the much better performance of Kenzo, is the object-oriented approach used in the former. The presence of object-oriented programming enables the reuse of data structures through inheritance, and the possibility of defining polymorphic operations (i.e., operations which can be applied to data of different, but related, types). In particular, these features are used in Kenzo to obtain a smart implementation of simplicial sets and chain complexes, structures which are chosen as paradigmatic examples in this paper.

In a series of papers, the data structures which appear in EAT have been analyzed [19, 18, 9, 17]. Nevertheless, the methods which have been used to deal with EAT cannot be directly applied to Kenzo, due mainly to the inheritance between data structures which appears in Kenzo.

Inheritance, like the notion of object-oriented as a whole, is a rather elusive concept [34]. Therefore its modelling by means of formal methods is a complex task, which can be approached from many different perspectives (see, for instance, [16, 13, 3, 7]). For our purpose in this paper, the specification side (and no the implementation) of inheritance is considered [29]. (On the contrary, in the work on EAT [19], implementation issues were the main point of interest.) Our approach is based on the hidden order-sorted specification framework (for order-sorted matters, see Ê[13], and for hidden ones [12]) but interpreting inheritance as a kind of *explicit coercion* [34, 4, 1].

The paper is organized as follows. Section 2 is devoted to some preliminaries on algebraic specifications. Section 3 explains the hidden specification of chain complexes, following the Kenzo style, and similarly, but more briefly, section 4 deals with simplicial sets. Syntactical aspects of inheritance between simplicial sets and chain complexes are tackled in section 5. Then two alternative ways for dealing with algebraic inheritance in the hidden context are explored in section 6. In section 7, we compare our approach with other works in Symbolic Computation. The paper ends with a section of conclusions and open problems.

2. PRELIMINARIES

We briefly introduce the basic notions on algebraic specifications and we refer to [20] for a systematic presentation.

In mathematics, when dealing with an algebraic structure, such as for instance a group, we refer to a set G together with some operations on G, $prd : G \times G \to G$, $inv : G \to G$, $unt :\to G$. This way of working is abstracted in the field of *universal algebra*, where structured-sets in this sense are studied in a generic way. Roughly speaking, algebraic specifications can be understood as universal algebra enriched with some syntactic constructs which establish a link between programming languages (through the notion of *type*) and mathematical structures.

More precisely, a *signature* Σ is a pair (S, Ω) of sets, whose elements are called *sorts* and *operations* respectively. Each

*Partially supported by DGES, project PB98-1621-C02-01 and by Universidad de La Rioja, project API-00/B28

ISSAC 2001, 7/01, Ontario, Canada
©2001 ACM 1-58113-417-7/ 01/ 0007 $5.00

operation consist of a $(k+2)$-tuple, $\omega : s_1 \ \ldots \ s_k \rightarrow s$ with $s_1, \ldots, s_k, s \in S$ and $k \geq 0$. In the case $k = 0$, the operation is called a *constant of sort s*. The sorts should be understood as the names for the sets to be defined, and the operations play the same role for operations on these sets. In the example of a group, the convenient signature is $S = \{g\}$ and $\Omega = \{* : g\ g \rightarrow g,\ ()^{-1} : g \rightarrow g,\ e :\rightarrow g\}$

Then the structures of universal algebra are retrieved by means of the notion of Σ-*algebra*. Let $\Sigma = (S, \Omega)$ be a signature. A *total algebra for* Σ (or Σ-*algebra*) assigns a set $A(s)$ to each sort $s \in S$, called the *carrier set* of the sort s, and a total function $A(\omega : s_1 \ \ldots \ s_k \rightarrow s) : A(s_1) \times \cdots \times A(s_k) \rightarrow A(s)$ to each operation $\omega \in \Omega$. In the example, we can define a Σ-algebra A taking $A(g) = G$, $A(*) = prd$, $A(()^{-1}) = inv$ and $A(e) = unt$.

The Σ-algebras can be organized as a category using the following natural notion of morphism. Let A, B be two Σ-algebras, with $\Sigma = (S, \Omega)$. A Σ-*homomorphism* $h : A \rightarrow B$ *from A to B* is a family $\{h_s : A(S) \rightarrow B(s)\}_{s \in S}$ of functions such that

$$h_s(A(\omega)(a_1, \ldots, a_k)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_k}(a_k))$$

for $\omega : s_1 \ \ldots \ s_k \rightarrow s \in \Omega$ and for all $a_i \in A(s_i)$, $i = 1, \ldots, k$.

When specifying *actual* programming systems, it is frequent to encounter maps which are *partial*, that is to say, which are undefined for certain arguments. A *partial algebra* [20, 2] A is defined as an algebra except that for each operation $\omega : s_1 \ \ldots \ s_k \rightarrow s$, the map $A(\omega) : A_{s_1} \times \cdots \times A_{s_k} \rightarrow A_s$ is defined on a (possibly proper) subset of $A_{s_1} \times \cdots \times A_{s_k}$, denoted by $Def(A(\omega))$.

In order to obtain a category of partial algebras, it is necessary to adapt the definition of homomorphism. One of the possibilities, which is useful for the purposes of this paper, is the following. A *weak homomorphism* between partial algebras [20, 2] is defined as a homomorphism $h : A \rightarrow B$ except that, for any operation $\omega : s_1 \ \ldots \ s_k \rightarrow s$, if $(a_1, \ldots, a_k) \in Def(A(\omega))$ then $(h_{s_1}(a_1), \ldots, h_{s_k}(a_k)) \in Def(B(\omega))$, and in this case: $h_s(A(\omega)(a_1, \ldots, a_k)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_k}(a_k))$.

Since we are interested in object-oriented matters, we will use a particular case of algebraic specification, known as *hidden specification* (see [12] for details).

Let $V\Sigma = (VS, V\Omega)$ be a signature. Let us fix a $V\Sigma$-algebra D and let us include in $V\Omega$, as constants, the elements of the carrier sets of D which do not correspond with constants previously found in $V\Omega$. The elements of VS are called *visible sorts* and those of $V\Omega$ are called *visible operations*. The $V\Sigma$-algebra D is called *data domain*. Then a *hidden signature*, on $V\Sigma$ and D, is a signature $H\Sigma = (S, \Omega)$ such that:

- $S = HS \sqcup VS$; the elements of HS are called *hidden sorts* of $H\Sigma$.
- $\Omega = H\Omega \sqcup V\Omega$ and for each operation $\omega : s_1, \ldots, s_k \rightarrow s$ in $H\Omega$ the following property holds: in $s_1, \ldots, s_k$ there is one and only one hidden sort and it is assumed this hidden sort appears in the first position (that is, it is s_1).

(This definition only covers a particular case of the notion introduced in [12], but it is enough for our purposes in this paper.)

A *hidden algebra* A for a hidden signature $H\Sigma$, on $V\Sigma$ and D, is a $H\Sigma$-algebra such that $A_{V\Sigma} = D$ (in other words, the restriction of A to the visible part is equal to the data domain D). A *hidden morphism* between two hidden algebras is a $H\Sigma$-homomorphism f such that f_D is the identity on D.

The (partial) hidden algebras for a hidden signature $H\Sigma$, on $V\Sigma$ and D, together with the (weak) hidden morphisms, define a category, which is denoted by $HAlg^D(H\Sigma)$.

3. HIDDEN SPECIFICATION OF CHAIN COMPLEXES

A *chain complex* $(C_p, d_p)_{p \in \mathbb{Z}}$ is a family of free $\mathbb{Z}$-modules $(C_p)_{p \in \mathbb{Z}}$, together with a family of $\mathbb{Z}$-module morphisms $(d_p)_{p \in \mathbb{Z}}$, the *differential maps*, such that $d_p : C_p \rightarrow C_{p-1}$, and $d_{p-1} \circ d_p = 0$, for each $p \in \mathbb{Z}$ (see [21]). The elements of C_p are called *combinations* of *degree p*.

Following closely the Kenzo way of working, a signature for dealing with the elements of chain complexes is composed of:

$$\begin{array}{rcl}
zero - cmbn & : & int \rightarrow cmbn \\
cmbn - opps & : & cmbn \rightarrow cmbn \\
n - cmbn & : & int \times cmbn \rightarrow cmbn \\
cmbn - degr & : & cmbn \rightarrow int \\
2cmbn - add & : & chcm \times cmbn \times cmbn \rightarrow cmbn \\
dffr & : & chcm \times cmbn \rightarrow cmbn \\
cmpr & : & chcm \times gnr \times gnr \rightarrow bool3
\end{array}$$

Here, the sort *gnr* stands for the *generators* set (in other words, the basis for any chain complex must be *subsets* of the carrier set for *gnr*), the sort *cmbn* for *combinations*, the sort *chcm* for the families of chain complexes, and *bool3* refers to a three-valued $(equal, greater, less)$ checking set. Thus, the meaning of the operations is clear. Let us only remark that *cmpr* represents a (partial) map which, at each degree, gives a total order on the basis of a concrete *chcm*.

From the hidden point of view, a hidden signature CC_{imp} (the suffix *imp* is used by the original relation of these constructions with implementation issues; see [19]) is obtained by declaring *chcm* as the unique hidden sort (this automatically classifies each operation as visible or hidden) and by defining a data domain D.

Let us define $D_{int} = \mathbb{Z}$, $D_{bool3} = \{equal, greater, less\}$ and $D_{gnr} = B$, where B is a graded set $B = \{B_p\}_{p \in \mathbb{Z}}$ (B will be the only variable set in the data domain). In order to define D_{cmbn} let us consider the signature formed by constants extracted from D_{int} and D_{gnr}, the operations $zero - cmbn$ and $cmbn - degr$ and a new operation:

$$add{-}mnm{-}to{-}cmbn : int \times gnr \times cmbn \rightarrow cmbn$$

This last operation is intended to capture the (partial) map formally adding a monomial to a combination (such an operation existed in EAT [27], but it appears in Kenzo only in a *destructive* version).

From this auxiliary signature, we complete a specification (see [20]), say AUX, by adding some natural axioms (for instance, $add-mnm-to-cmbn(0,a,x) = x$). Then we define D_{cmbn} as the initial model for this specification, an initial model which is described in the following result.

THEOREM 3.1. *A carrier set for cmbn in an initial algebra for AUX is*
$\{< p,[(t_1,a_1),(t_2,a_2),\ldots,(t_m,a_m)] > \mid p \in \mathbb{Z},\ m \in \mathbb{Z},$
$m \geq 0,\ t_i \in \mathbb{Z},\ t_i \neq 0$ *and* $a_i \in B_p,\ \forall i = 1,\ldots,m\}$

This description of the combinations in D_{cmbn} directly corresponds to the representation used in Kenzo.

This completes the definition of the hidden signature CC_{imp}, because visible operations are fixed in a natural way. Nevertheless in order to fit more closely the features of Kenzo, we introduce a syntactic construction which will be also useful when dealing with inheritance. We consider an operation:

$$coer-cmbn : int \times gnr \rightarrow cmbn$$

defined by: $coer-cmbn(p,a) =< p,[(1,a)] >$. When an operation symbol is prefixed by *coer*, this means that it is a *coercion* and then that certain previously defined operations are *overloaded* in a *polymorphic* way. In this particular case, it is assumed that a new operation

$$dffr : chcm \times int \times gnr \rightarrow cmbn$$

is canonically defined by:

$$dffr(cc,p,a) := dffr(cc, coer-cmbn(p,a))$$

This accurately models the implementation strategy used in Kenzo [28].

We pay now some attention to the way in which chain complexes are represented in Kenzo. Let us denote for C_{imp} the subcategory of $HAlg^D(CC_{imp})$ formed by the objects on which the necessary axioms to obtain *actual* chain complexes (imposing $d_{p-1} \circ d_p = 0$ and so on) hold. Only the elements of D_{cmbn} whose generators are ordered with respect to *cmpr* are considered. Due to this restriction on *cmbn*, the function $2cmbn-add$, which is determined from *cmpr* in a natural way, is implemented in Kenzo by an efficient algorithm on these ordered elements (see again [28]).

Let us consider the hidden CC_{imp}-algebra A^{can} such that the elements of A^{can}_{chcm} are pairs of functions (c,d), with $c : D_{gnr} \times D_{gnr} \rightarrow D_{bool3}$ and $d : D_{cmbn} \rightarrow D_{cmbn}$ such that c is a total order on the generators at each degree and d is a representation of the differential of a chain complex, both satisfying partiality conditions. Then it is straightforward to complete the definition of the CC_{imp}-algebra A^{can}. The following result can be easily proved.

THEOREM 3.2. *The CC_{imp}-algebra A^{can} is a final object in C_{imp}.*

This result should be compared, firstly, with a general result stated in [12], and, secondly, with the implementation strategy used in Kenzo [28]. The theorem on the existence of hidden final objects formally proves that the Kenzo representation is the "most general" possible (being, nevertheless quite efficient, since it is "minimal", in a certain sense, among all the isomorphic final objects in C_{imp}) and it shows that the hidden machinery is suitable to specify symbolic computation systems like Kenzo.

4. HIDDEN SPECIFICATION OF SIMPLICIAL SETS

In the previous section, we have shown how the hidden techniques correspond very nicely to the way of working in the Kenzo system. For simplicial sets, things are a bit more complex, because in Kenzo simplicial sets are considered a *subclass* of chain complexes. In this section we show how the description given in [27] for simplicial sets in the EAT system can be adapted to the hidden framework (we refer to [22] and [18] for the general definitions on simplicial sets).

The "minimal" signature for dealing with simplicial sets is:

$$\begin{array}{rcl} dgnr & : & nat \times absm \rightarrow absm \\ gmsm & : & absm \rightarrow gnr \\ face & : & smst \times nat \times nat \times absm \rightarrow absm \end{array}$$

where *gnr* denotes a set of *geometric simplexes*, *absm* the set of the *abstract simplexes* and *smst* is the *hidden* sort for simplicial sets. The operations *dgnr* and *face* represent, respectively, the degeneracy and face operators, and *gmsm* extracts from an abstract simplex the corresponding geometric simplex.

As data domain, we define $D_{nat} = \mathbb{N}$, $D_{gnr} = B = \{B_p\}_{p \in \mathbb{Z}}$, with $B_p = \emptyset$ if $p < 0$ (from this technical condition, we will get an homogeneous representation of geometric simplexes and chain complexes generators) and we choose as D_{absm}, the initial model for the following specification. The signature contains the elements of D_{nat} and D_{gnr} as constants, the operation *dgnr* and, in addition, a new operation:

$$coer-absm : gnr \rightarrow absm$$

intended to transform a geometric simplex into the corresponding *non-degenerate* abstract simplex. This signature, together with the natural axioms and partiality conditions, define a specification whose initial model is used to define D_{absm}, the set of abstract simplexes. A description for D_{absm} is:

$\{< (j_k,\ldots,j_1),a > \mid a \in B_n,$ for some $n \in \mathbb{N},\ k \in \mathbb{Z},$
$k \geq 0,\ j_i \in \mathbb{N},\ \forall i = 1,\ldots,k,\ j_k < n+k$ and $j_i > j_{i-1},$
$\forall i = 2,\ldots,k\}$

(This is not really the description suggested by Kenzo, which is based on a smart and very efficient numerical encoding of the *degeneracy list* $(j_k,\ldots,j_1)$, but it is closer to the usual presentation in simplicial topology; see [22].)

This completes the definition of a hidden signature for simplicial sets since the visible operations are fixed on the data domain. In particular we define: $coer-absm(a) :=< (),a >$

for each geometric simplex a, and, if this operation is included in the signature, a polymorphic operation (which is present in Kenzo) appears:

$$face : smst \times nat \times nat \times gnr \rightarrow absm$$

As for chain complexes, a final object for a hidden category is obtained by simply storing the tuples of functions associated with the algebraic structure. In this case the only essential function is the face operator:

$$f : D_{nat} \times D_{nat} \times D_{absm} \rightarrow D_{absm}$$

(see [18] for details).

5. PUTTING TOGETHER CHAIN COMPLEXES AND SIMPLICIAL SETS

As it has been previously mentioned, Kenzo considers a simplicial set as a particular case of a chain complex, because a simplicial set can be interpreted as an *Eilenberg-MacLane FD-complex* (see [22], page 93). Roughly speaking, a simplicial set $(X, face)$ is endowed with a differential structure

$$d_n : C_n(X) \rightarrow C_{n-1}(X)$$

where $C_n(X)$ is the free $\mathbb{Z}$-module generated on the n-geometric simplexes of X, and essentially,

$$d_n(-) := \sum_{i=0}^{n} (-1)^n face(X, i, n, -).$$

The previous system EAT provides a construction function which builds this chain complex from a simplicial set, when needed. Nevertheless, Kenzo adds to the simplicial set structure this particular chain complex using the inheritance technique provided by object-oriented programming, getting a reuse of the common elements. Besides, we obtain, in an automatic way, some important polymorphic functions, such as an equality test, *cmpr*, or the differential operation *diffr*. This illustrates, in this particular case, the benefits of the object-oriented programming from the software engineering point of view.

To deal with this new situation, a first attempt is to use an *order-sorted specification* [13]: a new signature SS_{imp} is obtained by adding to the signature CC_{imp} of section 3 the operations on simplicial sets of section 4, by repeating the operations of CC_{imp} changing everywhere the sort *chcm* by *smst*, and by declaring $smst < chcm$.

However, this approach is not convenient at least for two reasons. The first one is syntactical in nature: the *reuse* associated to the inheritance concept is lost, because some operations must be *explicitly* redefined (for overloading). The second one is most important: to the syntactical declaration $smst < chcm$ corresponds, at the model level, the fact that $A_{smst} \subseteq A_{chcm}$ for each *order-sorted* algebra (see [13]). But, the final objects in the previous sections illustrate the well-known fact that inheritance, in the *universal algebra* context, is rather a *forgetting* matter and not an *inclusion* one.

This weakness of the original order-sorted approach has been remarked by several authors. In particular in [23], in the context of the CoFI Algebraic Specification Language, CASL [30], two subsorting relationship $\leq^1, \leq^2$ are considered. The first one $\leq^1$ is related to the usual interpretation as inclusions, and the second one $\leq^2$ is closer to the interpretation as coercions. (The idea of interpreting a sort relation as a coercion is not original from [23], since it had previously been proposed by other authors in [34, 4], for instance.) Obviously our declaration $smst < chcm$ should be interpreted as $smst \leq^2 chcm$ rather than as $smst \leq^1 chcm$.

Thus, we define our definitive SS_{imp} signature by adding to CC_{imp} the operations on simplicial sets and a new operation:

$$coer - chcm : smst \rightarrow chcm$$

Even if this operation does not appear explicitly in Kenzo (it is subsumed by the fact that simplicial sets are defined as a subclass of chain complex[1]), it allows us to specify, at the syntactical level, all the Kenzo features (including polymorphism/overloading of operations) without including any redundant information.

From the semantical point of view, it is needed to require $coer - chcm$ to respect the equality test between geometric simplexes (let us observe that the geometric simplexes of the simplicial set are the generators of the associated chain complex) and the differential operator (that is to say, the differential associated to a coerced chain complex must be coherent with the corresponding as a FD-complex).

Bearing in mind these conditions, the hidden specification of inheritance is approached in two different ways in the following section.

6. HIDDEN SPECIFICATION OF INHERITANCE

Let us note that even if the signature SS_{imp} has been completely defined in the previous section, its nature as *hidden* signature has not been elucidated yet. Essentially it lacks the determination of the *hidden* sorts. It is natural to require the sort *smst* to be a hidden sort. But, the nature of *chcm* is more controversial.

If *smst* and *chcm* are considered "at the same level", they should be both hidden sorts. On the contrary, if *chcm* is considered a previous, auxiliary, data structure, it should be declared a visible sort. In the following two subsections, both alternatives are explored.

6.1 Hidden signature with two hidden sorts

If both *smst* and *chcm* are declared hidden sorts, the signature SS_{imp} can be directly considered a hidden signature, since the data domain parts D_{cmbn} and D_{absm} can be fixed by initial models as explained in sections 3 and 4.

Nevertheless, this hidden signature is more complex than those analyzed in [19], because it is not a pure *deconstructor* signature (see [17]). In other words,

$$coer - chcm : smst \rightarrow chcm$$

is a *constructor* for the hidden sort *chcm*, and this implies that the techniques to be used are more complex. But this

[1]To be precise, simplicial sets are a subclass of coalgebras and these are a subclass of chain complexes.

kind of signatures are covered by the result on the existence of final objects in hidden categories of [12]. This result is not directly applicable to our case, because we are dealing with *partial* algebras, but we are able to modify it to our particular case and to prove the following result.

THEOREM 6.1. *The hidden category of simplicial sets on the signature SS_{imp}, with two hidden sorts smst and chcm, has a final object.*

In addition, this final object admits the following *functional* description, denoted by B^{can}. The elements of B^{can}_{chcm} are the same pairs of functions (c, d) of the section 3, and the elements of B^{can}_{smst} are pairs of functions (c, f), with $c : D_{gnr} \times D_{gnr} \rightarrow D_{bool3}$ and $f : D_{nat} \times D_{nat} \times D_{absm} \rightarrow D_{absm}$ such that c is a total order on the geometric simplexes at each degree and f is a representation of the operator face of a simplicial set, both satisfying the natural partiality conditions. Now the constructor is defined in a natural way: $B^{can}(coer - chcm)(c, f) = (c, d_f)$, where d_f is a representation of the differential FD-operator defined from f. This final object corresponds very closely to the way in which simplicial sets have been implemented in the Kenzo system.

In order to interpret the operation $coer - chcm$, let us note that the explicit representation of simplicial sets is given by four maps (since visible operations are fixed on the data domain):

$$\begin{array}{rcl} c & : & D_{gnr} \times D_{gnr} \rightarrow D_{bool3} \\ f & : & D_{nat} \times D_{nat} \times D_{absm} \rightarrow D_{absm} \\ + & : & D_{cmbn} \times D_{cmbn} \rightarrow D_{cmbn} \\ d_f & : & D_{cmbn} \rightarrow D_{cmbn} \end{array}$$

But the addition is induced by the comparison test c, and the differential by c and the face operator f. Hence in the final object only c and f are necessary. If a simplicial set is identified with the four operations above, it is clear that the coercion $coer - chcm$ (and thus the inheritance relationship) can be interpreted as a *forgetful* mapping.

6.2 Hidden signature with a unique hidden sort

If we decide to declare *chcm* as a *visible* sort, then things are easier, because the signature SS_{imp} becomes a *deconstructor* signature and the general results for this particular kind of signatures of.[19] and [17] can be applied.

But in this case the data domain must be completed with a new set D_{chcm}. The elements in D_{chcm} should be interpreted as the ground on which simplicial sets are built. Bearing in mind this interpretation, it is clear that to get enough simplicial sets, D_{chcm} should be defined as the carrier set for the final object of Theorem 3.2. The fact that D_{cmbn} and D_{absm} are defined through initial models, while D_{chcm} is fixed by means of a final model, reflects the different nature of *cmbn* (or *absm*) and *chcm*, even being both visible sorts: the first one specifies *elements* (so, it is convenient to get as few data items as possible), and the second one specifies *families of elements* of the first type (and then we need a representation as general as possible).

On this hidden signature, the operation $coer - chcm$ can be not only constrained, but completely defined. The definition illustrates, again, the coercion/inheritance relationship as a forgetful mapping.

Then the next theorem follows from general results in [12] and [17].

THEOREM 6.2. *The hidden category of simplicial sets on the signature SS_{imp}, with a unique hidden sort smst, has a final object.*

Besides, the final object (as it is showed in [17]) can be described by means of tuples of functions. Interestingly enough (but not surprisingly), the functional final objects of the two last theorems are exactly identical as standard (no hidden) SS_{imp}-algebras (obviously the final morphisms are different in both categories). As a consequence, we deduce that this modelling technique also corresponds nicely to the Kenzo implementation strategy.

7. RELATED WORK IN SYMBOLIC COMPUTATION

In the previous sections we have mentioned, when necessary, references both to the theory of object-oriented programming and to the algebraic specifications field. In this section we focus on papers dealing with the interaction between Symbolic Computation and Type Theory.

The paper should be first read in the context of our previous papers on the EAT system: [19] (where implementation issues are mathematically modeled), [17] (in which the relationship with hidden specifications and coalgebras is explored) and [9] (where our constructions are expressed in an institutional framework; for the concept of *institution*, see [24] or [6]). Our approach seems to be quite original, in the sense that it deals with the *modeling* of a system already produced: we are not trying to explain the way in which a type system has been designed or used for implementing a computer algebra program, but rather to obtain mathematical models (through algebraic specifications machinery) in order to have enough resources for *reasoning* on the *internal* processes of computation in the EAT and Kenzo systems.

Nevertheless, it is clear that our research line is not separate from the main topics in type systems for Symbolic Computation. We briefly comment some of these topics.

In a first block of papers, we find those related to, or inspired by, the system AXIOM (previously Scratchpad) [15]. And the main reference should be the work explaining the ideas used in the development of AXIOM, [8] for instance. There are several differences between our approach and the one in AXIOM. A first source of differences comes from the systems themselves: AXIOM has been designed to be a general purpose Computer Algebra system and Kenzo is a special purpose program created for computing homology and homotopy groups. In addition, the research in [8] focused on a system in progress and the aim was, among others, to improve the type system; on the contrary our objective is not to influence the Kenzo system (which is running, and well, since several years ago), but rather to introduce tools for reasoning on its results. From a technical point of view,

AXIOM is based on a type system which is explicitly *second-order* (through the notions of *category* and *domain*), while Kenzo is directly constructed on CLOS and then the dynamic typing strategy of Common Lisp is used. We claim that, in order to specify EAT and Kenzo, the standard first-order approach is enough (this is the reason why we rely on [20] and [12], for instance, and no on higher-order techniques in algebraic specifications, such as [2] for example), even if the implementation uses (higher-order) functional programming intensively. Finally, a last difference between the work on AXIOM and our approach is the relevance of *infinite data structures*. Even if in AXIOM domains can be implicitly infinite, in EAT and Kenzo (and, in fact, in any *general* system for computing in Algebraic Topology) this feature must be explicitly managed. Or, rather, the unusual aspect is that both finite and infinite objects (*effective* and *locally effective* objects, in Sergeraert's terminology [26], [25]) must be considered: with the first ones we can compute (the Betti numbers of a *finite* simplicial set, for instance) and the second ones can be handled (by means of certain functors) and are used for computing with their *elements* (computing the faces of a simplex, for example; no difference with AXIOM on this second aspect).

The notion of *coercion* has been used by several authors in the field of Symbolic Computation, in particular by Weber [31], [32], [33] and Doye [11], [10]. Weber's approach is more "syntactic" in nature: he deals with the notion of coercion (and the related concept of *coherence*) in type systems for Computer Algebra packages and is interested in *type inference*, and not, as in our case, in the abstract data type, model-based, point of view. Doye's perspective is closer to ours, since he uses order-sorted algebras, but his aims are to prove the (general and algorithmic) *existence* of coercions for pairs of AXIOM types. In our case, a coercion is used to model, at the algebraic level, an already existing relationship at the implementation level between data structures: the relationship induced by inheritance in an object-oriented programming language (hence, both the existence and the algorithmic nature of the relation is *a priori* known).

Last in this block, the Weyl computer algebra substrate [35] is a system written, as Kenzo, in CLOS. The Weyl system provides an *infrastructure for developing* computer algebra programs embedded in more general applications (integrating numerical methods and user interfaces, for example). However, our objective is to give a *superstructure for reasoning* on some concrete systems. An interesting question (but quite unrelated to our research project) is to know whether the Kenzo system could be suitably reprogrammed on the Weyl substrate. Another problem, closer to our perspective, is to study whether our techniques can be, more o less directly, applied to the programs written on Weyl.

A second source of references on these topics is due to Calmet et al. [6], [5], [14]. These papers deal with the problem of *knowledge representation* and, concretely, with the representation of mathematical knowledge by means of algebraic structures. The formalism used for the specifications in the language FORMAL [6] is that of *unified algebras* [24], which allows the analyst to calculate in an integrated way with the elements and with the sorts of a specification. This point of view could be used as an alternative approach to model the *two-layer* organization of EAT and Kenzo data structures: the computation with algebraic structures and the computation with the *elements* of the algebraic structures (these two layers are also explicitly present in Weyl [35], through the notions of *domain* and *domain element*, and implicitly in AXIOM [15]). More work will be necessary to know if the approaches of [6] and of this paper can be formally integrated.

8. CONCLUSIONS AND FURTHER WORK

This paper is a first attempt to understand the inheritance mechanisms used in a symbolic computation system, known as Kenzo. With this objective in mind, two alternative approaches based on hidden specifications and coercions have been explored. We have focused on the particular case of simplicial sets and chain complexes. On these particular structures both approaches seem suitable and it is difficult to decide which is the right one (if any). The first idea (to declare both sorts involved in inheritance as hidden sorts) seems more natural, but implies more technical difficulties. Further work will be necessary to elucidate this point, moving from particular cases to a general setting. If finally these ideas are considered fruitful, then it will be necessary to translate them from the *specification inheritance* to the *implementation inheritance* field, since our actual interest is to explain, as close as possible and in a formal way, the object-oriented features of the Kenzo system. In addition, the questions raised in the previous section (in particular, the relationships with Weyl [35] and FORMAL [6]) require further investigation.

9. ACKNOWLEDGMENTS

We thank to the anonymous referees several interesting pointers to related literature.

10. REFERENCES

[1] BREAZU-TANNEN, V., COQUAND, T., GUNTER, C., AND SCEDROV, A. Inheritance as explicit coercion. *Information and Computation 93* (1991), 172–221.

[2] BROY, M. Equational specification of partial higher-order algebras. *Theoretical Computer Science 57* (1988), 3–45.

[3] BRUCE, K. The equivalence of two semantic definitions for inheritance in object-oriented languages. In *Mathematical Foundations of Programming Semantics* (1991), Pittsburgh, pp. 102–124.

[4] BRUCE, K., AND WEGNER, P. An algebraic model of subtype and inheritance. In *Advances in Database Programming Language* (1990), Addison-Wesley, pp. 75–96.

[5] CALMET, J., HOMANN, K., AND TJANDRA, I. A. Unified domains and abstract computational structures. In *Proceedings AISMC'93* (1993), Lecture Notes in Computer Science 737, pp. 166–177.

[6] CALMET, J., AND TJANDRA, I. A. A unified-algebra-based specification language for symbolic computing. In *Proceedings DISCO'93* (1993), Lecture Notes in Computer Science 722, pp. 122–133.

[7] CARDELLI, L. A semantics of multiple inheritance. *Information and Computation 76* (1988), 138–164.

[8] DAVENPORT, J. H., AND TRAGER, B. M. Scratchpad's view of algebra I: Basic commutative algebra. In *Proceedings DISCO'90* (1990), Lecture Notes in Computer Science 429, pp. 40–54.

[9] DOMÍNGUEZ, C., LAMBÁN, L., PASCUAL, V., AND RUBIO, J. Hidden specification of a functional system. In *Proceedings EUROCAST'2001* (2001), Universidad de Las Palmas de Gran Canaria.

[10] DOYE, N. *Order Sorted Computer Algebra and Coercions.* PhD thesis, University of Bath, 1997.

[11] DOYE, N. Automated coercion for AXIOM. In *Proceedings ISSAC'99* (1999), ACM Press, pp. 229–235.

[12] GOGUEN, J., AND MALCOLM, G. A hidden agenda. *Theoretical Computer Science 245* (2000), 55–101.

[13] GOGUEN, J., AND MESEGUER, J. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science 105* (1992), 217–273.

[14] HOMANN, K., AND CALMET, J. Combining theorem proving and symbolic mathematical computing. In *Proceedings AISMC'95* (1995), Lecture Notes in Computer Science 958, pp. 18–29.

[15] JENKS, R. D., AND SUTOR, R. S. *AXIOM: The Scientific Computation System.* Springer-Verlag, 1992.

[16] KAMIN, S., AND REDDY, U. Two semantic models of object-oriented languages. In *Theorical Aspects of Object-Oriented Programming* (1994), MIT Press, pp. 463–495.

[17] LAMBÁN, L., PASCUAL, V., AND RUBIO, J. An object-oriented interpretation of the EAT system. Preprint.

[18] LAMBÁN, L., PASCUAL, V., AND RUBIO, J. Simplicial sets in the EAT system. In *Proceedings EACA'99* (1999), Universidad de La Laguna, pp. 267–276.

[19] LAMBÁN, L., PASCUAL, V., AND RUBIO, J. Specifying implementations. In *Proceedings ISSAC'99* (1999), ACM Press, pp. 245–251.

[20] LOECKX, J., EHRICH, H. D., AND WOLF, M. *Specification of Abstract Data Types.* Wiley-Teubner, 1996.

[21] MAC LANE, S. *Homology.* Springer-Verlag, 1975.

[22] MAY, J. P. *Simplicial Objects in Algebraic Topology.* Van Nostrand, 1967.

[23] MOSSAKOWSKI, T., HAXTHAUSEN, A., AND KRIEG-BRÜCKNER, B. Subsorted partial higher-order logic as an extension of CASL. In *Proceedings WADT'99* (2000), Lecture Notes in Computer Science 1827, pp. 126–145.

[24] MOSSES, P. D. Unified algebras and institutions. In *Logics in Computer Science* (1989), IEEE Press, pp. 304–312.

[25] RUBIO, J. Locally effective objects and artificial intelligence. In *Proceedings AISC'2000* (2000), Lecture Notes in Artificial Intelligence 1930.

[26] RUBIO, J., AND SERGERAERT, F. Locally effective objects and algebraic topology. In *Computational Algebraic Geometry* (1993), Birkhäuser, pp. 235–251.

[27] RUBIO, J., SERGERAERT, F., AND SIRET, Y. *EAT: Symbolic Software for Effective Homology Computation.* Institut Fourier, Grenoble, 1997. `ftp://fourier.ujf-grenoble.fr/pub/EAT`.

[28] SERGERAERT, F., AND SIRET, Y. *Kenzo: Symbolic Software for Effective Homology Computation.* Institut Fourier, Grenoble, 1999. `ftp://fourier.ujf-grenoble.fr/pub/KENZO`.

[29] SNYDER, A. Inheritance and the development of encapsulated software components. In *Research Directions in Object-Oriented Programming* (1987), MIT Press, pp. 165–188.

[30] THE COFI TASK GROUP ON LANGUAGE DESIGN. CASL, The Common Algebraic Specification Language - Summary. Version 1.0. Tech. rep., 1999. `http://www.brics.dk/Projects/CoFI/Documents/CASL/Summary/`.

[31] WEBER, A. A type-coercion problem in computer algebra. In *Proceedings AISMC'92* (1992), Lecture Notes in Computer Science 737, pp. 188–194.

[32] WEBER, A. Algorithms for type inference with coercions. In *Proceedings ISSAC'94* (1994), ACM Press, pp. 324–329.

[33] WEBER, A. On coherence in computer algebra. *Journal of Symbolic Computation 19* (1995), 25–38.

[34] WEGNER, P. The object-oriented classification paradigm. In *Research Directions in Object-Oriented Programming* (1987), MIT Press, pp. 479–560.

[35] ZIPPEL, R. The Weyl computer algebra substrate. In *Proceedings DISCO'93* (1993), Lecture Notes in Computer Science 722, pp. 303–318.