Journal of Universal Computer Science, vol. 12, no. 12 (2006), 1701-1730
submitted: 17/10/06, accepted: 23/12/06, appeared: 28/12/06 © J.UCS

Modeling Inheritance as Coercion in the Kenzo System

César Domínguez
(Universidad de La Rioja, Spain
cesar.dominguez@unirioja.es)

Julio Rubio
(Universidad de La Rioja, Spain
julio.rubio@unirioja.es)

Francis Sergeraert
(Institut Fourier, Université Grenoble I, France
francis.sergeraert@ujf-grenoble.fr)

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 show how the order-sorted algebraic specification formalism can be adapted, through the "inheritance as coercion" metaphor, in order to model the simple inheritance between structures in Kenzo.
Key Words: Algebraic specification, symbolic computation, inheritance, coercion.
Category: F.3.1

1 Introduction

Kenzo is a system designed by the third author [Dousson et al. 1999] for the calculation of homology and homotopy groups for topological spaces. It is written in CLOS and is a descendant of the first system developed by the third author for symbolic computation in Algebraic Topology, called EAT [Rubio et al. 1997].

The main difference between Kenzo and EAT is the object-oriented approach of the former, as well as its superior performance. The presence of object-oriented programming enables the reuse of data structures through inheritance, and the possibility of defining polymorphic operations (i.e., operations that can be applied to data of different, but related, types).

In a series of papers [Lambán et al. 1999b, Lambán et al. 1999a, Domínguez et al. 2001, Lambán et al. 2003, Domínguez et al. to appear], the data structures which appear in EAT have been analyzed. Nevertheless, the methods used to deal with EAT cannot be directly applied to Kenzo, mainly due 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 [Weber 1987, Taivalsaar 1996]. Therefore its modeling by means of formal methods is a complex task, which can be approached from many different perspectives (see, for instance, [Kamir and Reddy 1994, Goguen and Meseguer 1992, Bruce 1992, Cardelli 1988]). To our aim in this paper, the specification side (and not the implementation) of inheritance is considered [Snyder 1987]. (On the contrary, in the work on EAT [Lambán et al. 1999b], implementation issues were the main point of interest.) Our approach is based on the hidden order-sorted specification framework (for order-sorted matters, see [Goguen and Meseguer 1992], and for hidden ones [Goguen and Malcolm 2000]) but interpreting inheritance as a kind of *explicit coercion* [Weber 1987, Bruce and Wegner 1990, Breazu-Tannen et al. 1991].

The technique explained in this paper was previously developed in [Domínguez and Rubio 2001] for the specification of a particular example: the chain complex/simplicial set relationship. In this work, it is generalized to include all the simple inheritance relations between structures in Kenzo.

The paper is organized as follows. Section 2 deals with some preliminaries on algebraic specifications. Section 3 includes an introductory example. Section 4 explains the hidden specification of structures in EAT, through an operation called *imp* operation, and Section 5 provides some examples of this specification. This technique is not valid for Kenzo structures due to the presence of inheritance in their construction. So, Section 6 deals with the specification of the simple inheritance between structures in Kenzo. First, the syntactical aspects of inheritance are tackled, and then, two alternative approaches to deal with algebraic inheritance in the hidden context are explored. Section 7 presents some examples of this technique. In Section 8, we compare our approach with other related works. Section 9 presents the conclusions and includes some open problems. The paper ends with an appendix devoted to a short introduction to the Kenzo system that tries to reflect some of the characteristics previously studied.

2 Preliminaries on algebraic specification

We will briefly introduce some basic notions on algebraic specifications; see [Loeckx et al. 1996] 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, $*\colon G \times G \to G$, $-\colon G \to G$, $e\colon\ \to G$. 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 that 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 operation consists of a $(n+2)$-

tuple, $\omega\colon s_1 \dots s_n \to s$ with $s_1, \dots, s_n, s \in S$ and $n \geq 0$. In the case $n = 0$, 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, denoted by GRP, has one sort g and three operations $prd\colon g\ g \to g$, $inv\colon g \to g$, $unt\colon\ \to g$.

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 $\omega_A\colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ to each operation $\omega\colon s_1 \dots s_n \to s \in \Omega$. In the example, we can define a Σ-algebra A taking $A_g = G$, $prd_A = *$, $inv_A = -$ and $unt_A = e$.

The Σ-algebras can be organized as a category $Alg(\Sigma)$ using the following natural notion of morphism. Let A, B be two Σ-algebras, with $\Sigma = (S, \Omega)$. A *Σ-homomorphism $h\colon A \to B$ from A to B* is a family $\{h_s\colon A_s \to B_s\}_{s \in S}$ of functions such that $h_s(\omega_A(a_1, \dots, a_n)) = \omega_B(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$ for any operation $\omega\colon s_1 \dots s_n \to s \in \Omega$ and for all $a_i \in A_{s_i},\ i = 1, \dots, n$.

In the specification of *actual* programming systems, it is usual to find *partial* maps, that is to say, maps which are undefined for certain arguments. A *partial algebra* [Loeckx et al. 1996, Cerioli et al. 1999] A is defined as an algebra except that for each operation $\omega\colon s_1 \dots s_n \to s$, the map $\omega_A\colon A_{s_1} \times \cdots \times A_{s_n} \to A_s$ is defined on a (possibly proper) subset of $A_{s_1} \times \cdots \times A_{s_n}$, denoted by $Def(\omega_A)$.

In order to obtain a category of partial algebras $PAlg(\Sigma)$, 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 [Loeckx et al. 1996] is defined as a homomorphism $h\colon A \to B$ except that, for any operation $\omega\colon s_1 \dots s_n \to s$, if $(a_1, \dots, a_n) \in Def(\omega_A)$ then $(h_{s_1}(a_1), \dots, h_{s_n}(a_n)) \in Def(\omega_B)$, and in this case: $h_s(\omega_A(a_1, \dots, a_n)) = \omega_B(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$.

In this work, we will use the notion of *quotient algebra*, which is an algebra in which some carriers are identified with each other. The identification is obtained with the help of a *congruence relation*. In particular, we need the partial version of this algebra [Cerioli et al. 1999]. Let $\Sigma = (S, \Omega)$ be a signature and A be a Σ-algebra. A family $Q = (Q_s)_{s \in S}$ of *partial equivalence relations* (symmetric and transitive relations), Q_s on A_s, is a *partial congruence relation on A* if for each operation $\omega\colon s_1 \dots s_n \to s \in \Omega$, and for any tuples $(a_1, \dots, a_n), (b_1, \dots, b_n) \in Def(\omega_A)$ such that $a_i\ Q_{s_i}\ b_i$, $i = 1, \dots, n$, then $\omega_A(a_1, \dots, a_n)\ Q_s\ \omega_B(b_1, \dots, b_n)$. Given a partial congruence relation on the Σ-algebra A, the *partial quotient algebra of A by Q* is the Σ-algebra $A/_Q$ defined by: $(A/_Q)_s = \{[a]_{Q_s} \mid a\ Q_s\ a\}$ for each $s \in S$, and $\omega_{A/_Q}([a_1]_{Q_{s_1}}, \dots, [a_n]_{Q_{s_n}}) = [\omega_A(x_1, \dots, x_n)]_{Q_s}$, for each $\omega\colon s_1 \dots s_n \to s \in \Omega$, where $([a_1]_{Q_{s_1}}, \ldots, [a_n]_{Q_{s_n}}) \in Def(\omega_{A/Q})$ iff exists $x_i \in [a_i]_{Q_{s_i}}$, $i = 1, \ldots, n$ such that $(x_1, \ldots, x_n) \in Def(\omega_A)$.

Since we are interested in object-oriented matters, we will use a particular case of algebraic specification, known as *hidden specification* (see [Goguen and Malcolm 2000] 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 that do not correspond with constants previously found in $V\Omega$. This new set is also named $V\Omega$ and it is used instead of the previous one. 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 $\omega\colon s_1 \ldots s_n \to s$ in $H\Omega$ the following property holds: in $s_1, \ldots, s_n$ there is at most one hidden sort.

(The $\sqcup$ symbol denotes the disjoint union.)

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 h such that h_{VS} 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 $HPAlg^D(H\Sigma)$.

3 An introductory example: families of groups

As an introductory example let us consider the simple case of groups. Let us denote by GRP the category of groups. This corresponds to a subcategory of total algebras of the signature $\mathtt{GRP}$ described in the previous section, satisfying in addition the well-known axioms for groups.

Let us denote now by $GRP^{\mathbb{Z}}$ the category of groups built over a fixed underlying set $\mathbb{Z}$. This is to say, that the carrier set for a group is $\mathbb{Z}$ or, more generally, a quotient of $\mathbb{Z}$. (This new category appears in a natural way when modeling a programming scenario: $\mathbb{Z}$ being an example of a type in a programming language.) Let us define a *family of groups* as a function $G\colon A \to Obj(GRP^{\mathbb{Z}})$, where A is a set, called *index set*, and $Obj(GRP^{\mathbb{Z}})$ denotes the class of objects in the category $GRP^{\mathbb{Z}}$. An example is the family of all groups module n, where $A = \mathbb{N}$ and $G(n) = \mathbb{Z}/n\mathbb{Z}$, for each $n \in \mathbb{N}$. These families are objects of a category denoted by $HPAlg^{\mathbb{Z}}(\mathtt{GRP}^{eq}_{imp})$. This category will be our main entity of study and the reasons for this notation will be explained later on. The main feature of this category is it has a final object. More precisely, the elements of the index set of the final object are tuples of functions defining a group over (a quotient of) $\mathbb{Z}$. This object nicely corresponds to the implementation of the data structures used in Kenzo.

4 The *imp* operation

Before trying to obtain a specification for inheritance relationships between structures in Kenzo, we have to consider the specification of a simple structure where inheritance is not used in its construction. This situation is similar to the one presented for the EAT structures in [Lambán et al. 1999b]. In that work, an operation called *imp* was defined to obtain the specification of those structures. In this section, we present the same operation with some slight differences.

The initial data is a category, or rather a class of objects $\mathcal{T}$, we want to model (the class of groups, the class of simplicial sets and so on). Then, a signature $\Sigma = (S, \Omega)$ is designed with the aim *of obtaining* Σ-algebras which represent (in some sense) the objects in $\mathcal{T}$. Nevertheless, it is not sensible to think of representing *any* object in $\mathcal{T}$ (any group, for example), both from a computability point of view (usually, $\mathcal{T}$ is a non-numerable class) and from a practical one (it is very useful to identify the ground elements of some sort with a unique programming *type*). This restriction leads, at the model level, to the need of fixing a data domain $D = \{D_s\}_{s \in S}$ for the signature Σ. The idea is to consider only Σ-algebras with carrier sets in D.

The apparent strictness of working with a fixed data domain could be overcome in at least two ways. On the one hand, the models can be defined on *quotient sets* of the data domain D (and not on D itself). To illustrate this point, let us consider the case of the groups. If $D = \mathbb{Z}$, the set of integer numbers, we are dealing with groups represented on $\mathbb{Z}$. So, a finite group such as $\mathbb{Z}/n\mathbb{Z}$ can be represented on D if equalities (i.e. equivalence relations) on $\mathbb{Z}$ are allowed. On the other hand, the models can have *partial* operations. Then, for example, the semigroup $\mathbb{N}$ can be represented on the data domain $D = \mathbb{Z}$ if we consider the operation of the semigroup partial with the definition domain on $\mathbb{N}$. In particular, in the field of Algebraic Topology, structures are based on *graded* sets (usually, the degree is related to some notion of geometric dimension), and this implies that operations are only defined for elements of same degree; see the particular cases of simplicial sets and chain complexes below (we refer to [May 1982] for the mathematical definition of these structures).

In our context, we allow for both aspects, although working with a fixed data domain, we can obtain different domains either because we define a quotient on the data domain or because, through partiality, we only take into account one data domain subset.

Given a signature $\Sigma = (S, \Omega)$ and a data domain D fixed for it, to deal with the above points we need the following process. The signature Σ is enriched with a boolean test operation $eq^s \colon s\ s \rightarrow \mathit{bool}$ for each $s \in S$, and a new signature, denoted by Σ^{eq}, is obtained. Then, for each Σ^{eq}-operation $\omega : s_1 \dots s_n \rightarrow s$ we fix a set $dom_\omega \subseteq D_{s_1} \times \cdots \times D_{s_n}$, the *definition domain* of ω, and we write $dom = \{dom_\omega\}_{\omega \in \Omega^{eq}}$. Therefore, we are going to work with the following category $PAlg^{D,dom}(\Sigma^{eq})$, that is to say, the category of partial Σ^{eq}-algebras on D with definition domain *dom* and endowed with *weak morphisms* of partial algebras [Loeckx et al. 1996]. The fixing of the definition domain is motivated at the implementation level. Note that at the algebraic level every operation in an algebra is endowed with its definition domain, when it is defined as a partial function. So, it is not necessary to fix one of them and it is possible to obtain our results without this restriction. Nevertheless, we prefer model these structures as closely as possible to the Kenzo system, using examples directly extracted from this program. At the implementation level, for each operation of the signature there is only one code (that implements this operation) and a data domain where the code is syntactically correct, but, in general its real definition domain is not explicitly encoded. In fact, a given code can have different definitions domains. Thus, we decide to fix one of them in our modeling framework.

The category of models $\mathcal{C}$ considered is a full subcategory of $PAlg^{D,dom}(\Sigma^{eq})$ such that a Σ^{eq}-algebra $A \in \mathcal{C}$ satisfies two conditions. First, the family $eq_A = \{eq_s\}_{s \in S^{eq}}$ defines a partial congruence on A. Second, the quotient $A/_{eq_A}$ defines an object of $\mathcal{T}$.

In Kenzo and EAT, and in general in any symbolic computation system, you do not work only with a unique data structure, a group for example; in contrast, you deal at runtime with families of this data structure. So, we can distinguish two *layers* of data structures in these systems. In the first layer, ones finds the usual data structures as integer numbers or (finite) list of symbols. In the second layer, one must deal with algebraic structures as groups or chain complexes whose elements are data belonging to the first layer. To model this situation, in [Lambán et al. 1999b] an operation between specification frames, called *imp* operation, was defined. Here, we have modified slightly that operation to adapt it for our situation.

This operation assigns to each signature $\Sigma^{eq} = (S^{eq}, \Omega^{eq})$ with data domain D and definition domain *dom* a hidden signature $\Sigma^{eq}_{imp} = (S^{eq}_{imp}, \Omega^{eq}_{imp})$. This signature presents a new sort, denoted by $imp_{\Sigma^{eq}}$, and an operation $imp\_\omega \colon imp_{\Sigma^{eq}} s_1 \dots s_n \rightarrow s$ for each operation $\omega \colon s_1 \dots s_n \rightarrow s \in \Omega^{eq}$. The only hidden sort in this signature is the new sort and the data domain is D.

The category that we use for that signature is $HPAlg^{D,dom}(\Sigma^{eq}_{imp})$. This category has as objects the *partial* hidden Σ^{eq}_{imp}-algebras A on D and each operation $imp\_\omega \colon imp_{\Sigma^{eq}} s_1 \dots s_n \rightarrow s$ has as definition domain: $A_{imp_{\Sigma^{eq}}} \times dom_\omega$ (in other words, it will be total on the hidden argument). The morphisms of this category are the *weak* hidden Σ^{eq}_{imp}-homomorphisms.

We are interested in a full subcategory $\mathcal{C}_{imp}$ of $HPAlg^{D,dom}(\Sigma^{eq}_{imp})$ whose objects represent families of algebras in $\mathcal{C}$ (so, of $\mathcal{T}$-objects). In order to fulfill that idea, the distinguished sort will be used as index of these algebras, i.e. each element of the distinguished sort can be understood as a parameter allowing to define an algebra in $\mathcal{C}$. Hence, for each element $a \in A_{imp_{\Sigma^{eq}}}$, we must note that we can define a partial Σ^{eq}-algebra A^a. This algebra has the carrier set $A^a_s := D_s$ for each sort $s \in S^{eq}$, and the partial function $\omega_{A^a}(d_1, \ldots, d_n) := imp\_\omega_A(a, d_1, \ldots, d_n)$, with definition domain $Def(\omega_{A^a}) := dom_\omega$, for each operation $\omega\colon s_1 \ldots s_n \to s \in \Omega^{eq}$. Now, the algebras A in $\mathcal{C}_{imp}$ must verify that $A^a \in \mathcal{C}$ for each $a \in A_{imp_{\Sigma^{eq}}}$. So, we can distinguish the two layers of data structures in the algebra. The visible sorts represent the first layer: fixed data that are used to build the algebraic structures of the second layer, which is represented by the hidden sort.

Under these conditions, the category $\mathcal{C}_{imp}$ has a canonical object, which is denoted by A^{can}. This object can be presented as a set of partial functional tuples in the carrier set for the hidden sort (it is similar to the tuples of methods in Cardelli's approach [Abadi and Cardelli 1996]). More concretely, $A^{can}_{imp_{\Sigma^{eq}}} := \{(f_\omega)_{\omega\in\Omega^{eq}} \mid \langle D, (f_\omega, dom_\omega)_{\omega\in\Omega^{eq}}\rangle \in \mathcal{C}\}$ and $A^{can}_s := D_s$ for each sort $s \in S^{eq}$. Then, the partial functions of this algebra are defined in the natural way: $imp\_\omega_{A^{can}}((f_\delta)_{\delta\in\Omega^{eq}}, d_1, \ldots, d_n) := f_\omega(d_1, \ldots, d_n)$, for each $imp\_\omega\colon imp_{\Sigma^{eq}}\ s_1 \ldots s_n \to s \in \Omega^{eq}$, each $(f_\delta)_{\delta\in\Omega^{eq}} \in A^{can}_{imp_{\Sigma^{eq}}}$ and each tuple $(d_1, \ldots, d_n) \in dom_\omega$. Besides, this canonical object is a final object in $\mathcal{C}_{imp}$. This result is reflected in the following theorem.

Theorem 1. *The canonical object A^{can} is a final object in $\mathcal{C}_{imp}$.*

Proof. For each object $B \in \mathcal{C}_{imp}$, it is possible to define a weak hidden Σ^{eq}_{imp}-homomorphism, h^{can}, which has as component for the distinguished sort the total function $h^{can}_{imp_{\Sigma^{eq}}} : B_{imp_{\Sigma^{eq}}} \to A^{can}_{imp_{\Sigma^{eq}}}$, such that $h^{can}_{imp_{\Sigma^{eq}}}(b) := (\omega_{B^b})_{\omega\in\Omega^{eq}}$ for each $b \in B_{imp_{\Sigma^{eq}}}$. Besides, it is easy to prove that this homomorphism is unique.

This result should be compared, first, with the general result for hidden algebras reported in [Goguen and Malcolm 2000], and, second, with the implementation strategy used in Kenzo and EAT (see the appendix and the examples described below). The above theorem on the existence of hidden final objects formally proves that the representation chosen in these programs for their structures is the "most general" possible one (in the sense that exists a homomorphism between any other representation and the one chosen for this programs). It shows that the hidden machinery is suitable for specifying symbolic computation systems like Kenzo and EAT.

5 Examples of hidden specification of families of structures

In this section, we put into practice the specification technique previously defined with some families of structures: a family of groups over $\mathbb{Z}$ in a total context and a family of chain complexes and simplicial sets over a data domain in a general context. Then, we introduce the different treatment in the Kenzo system of a family of simplicial sets using inheritance techniques.

5.1 Families of groups revisited

The signature $HPAlg^D(\mathtt{GRP}^{eq}_{imp})$ introduced in Section 3 is obtained by applying the *imp* operation to the signature $\mathtt{GRP}^{eq}$, which is the signature for a group enriched with a boolean test operation on g. Besides, the set $D_g = \mathbb{Z}$ is fixed as data domain for the sort g. Then, we consider the subcategory of total algebras of $HPAlg^D(\mathtt{GRP}^{eq}_{imp})$ which satisfy that the quotient defined through the equality in the $\mathtt{GRP}^{eq}$-algebra is a group (i.e. the group axioms are satisfied over it). As in Section 3, the family of finite groups $\mathbb{Z}/n\mathbb{Z}$, $n \in \mathbb{N}$, is obtained as a hidden algebra with hidden carrier set $\mathbb{N}$ and the natural equality for every fixed $n \in \mathbb{N}$.

5.2 Families 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 $d_p\colon C_p \to C_{p-1}$, and $d_{p-1} \circ d_p = 0$, for each $p \in \mathbb{Z}$ (see [Mac Lane 1994]). The elements of C_p are called *combinations* of *degree p*.

Following the style of EAT and Kenzo, a signature $\mathtt{CC}_{imp}$ capable of dealing with the elements of chain complexes is composed of:

$$\begin{array}{rcl}
zero-cmbn & : & int \to cmbn \\
cmbn-opps & : & cmbn \to cmbn \\
n-cmbn & : & int\ cmbn \to cmbn \\
2cmbn-add & : & chcm\ cmbn\ cmbn \to cmbn \\
dffr & : & chcm\ cmbn \to cmbn \\
eq^{gnr} & : & chcm\ gnr\ gnr \to bool \\
eq^{cmbn} & : & chcm\ cmbn\ cmbn \to bool
\end{array}$$

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* and the sort *chcm* for the families of chain complexes. Thus, the meaning of the operations is clear.

Note that this signature is a simplification of the signature $\Sigma^{\mathtt{CC},eq}_{imp}$ obtained, through the *imp* operation, from the signature $\Sigma^{\mathtt{CC},eq}$ for a chain complex. This signature has the operations of $\mathtt{CC}_{imp}$ without the *chcm* argument and with an equality on the *int* sort. If we are dealing with free $\mathbb{Z}$-modules, we can canonically define the unit, inverse and product operations, so that they are the same for each chain complex and we can remove the first argument in these operations. The equality on the *int* sort will be considered as the literal equality on the integers.

As data domain we define $D^{cc}_{int} = \mathbb{Z}$ and $D^{cc}_{gnr} = B$, 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^{cc}_{cmbn} let us consider the signature formed by constants extracted from D^{cc}_{int} and D^{cc}_{gnr}, the operation $zero-cmbn$ and a new operation:

$$add{-}mnm{-}to{-}cmbn\colon int\ gnr\ cmbn \to cmbn$$

This operation is intended to capture the (partial) map that formally adds a monomial to a combination (such an operation exists in EAT and Kenzo). Then, we define D^{cc}_{cmbn} as the initial model for the category of algebras for this auxiliary signature. This model can be described as follows:

$$\{\langle p, [(t_1,a_1),(t_2,a_2),\ldots,(t_m,a_m)]\rangle \mid p\in\mathbb{Z},\ m\in\mathbb{Z},\ m\geq 0,\ t_i\in\mathbb{Z},\ \text{and}\ a_i\in B_p,\ \forall i=1,\ldots,m\}$$

This description of the combinations in D^{cc}_{cmbn} directly corresponds to the representation used in Kenzo.

This completes the definition of the hidden signature $\mathtt{CC}_{imp}$, because visible operations are fixed in a natural way. Now, we will rely on the partiality of the operations to obtain the graded structure: the addition and equalities are only defined on elements of the same degree.

Then, we consider the subcategory $\mathcal{C}^{cc}_{imp}$ of $HPAlg^{D^{cc},dom^{cc}}(\mathtt{CC}_{imp})$. An algebra A in this category satisfies that for every fixed element $a\in A_{chcm}$, its functions with this element fixed allow to define a quotient partial algebra that verifies the properties necessary to obtain *actual* chain complexes (imposing $d_{p-1}\circ d_p = 0$ and so on).

Now, in the subcategory $\mathcal{C}^{cc}_{imp}$ we can define a final object. In this case, the carrier set of the hidden sort is composed of tuples of four functions (visible operations have fixed functions and are not necessary to determine the chain complex). But, due to the identification produced during the quotient, the function $2cmbn-add$ can be determined from $cmpr$ in a natural way. Besides, the equality between combinations can be defined through the equality between generators. Then, only two functions $(d, =_{gnr})$ are needed. These functions are $d\colon D^{cc}_{cmbn}\to D^{cc}_{cmbn}$ and $=_{gnr}\colon D^{cc}_{gnr}\times D^{cc}_{gnr}\to D^{cc}_{bool}$, where d is a representation of the differential of a chain complex and $=_{gnr}$ is an equality on the generators at each degree, both satisfying partiality conditions.

In order to fit more closely the features of Kenzo, we can introduce a syntactic construction that will be also useful when dealing with inheritance. We consider a partial operation:

$$coer^{int\times gnr}_{cmbn}\colon int\ gnr\to cmbn$$

defined by: $coer^{int\times gnr}_{cmbn}(p,a) = \langle p, [(1,a)]\rangle$ on the tuples (p,a) such that $a \in B_p$. 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 partial operation

$$dffr\colon chcm\ int\ gnr \to cmbn$$

is canonically defined by:

$$dffr(cc,p,a) := dffr(cc, coer^{int\times gnr}_{cmbn}(p,a))$$

Therefore, this differential on generators determines the differential on combinations by linearity. So, this differential can replace the other one in the final object. This models accurately the implementation strategy used in Kenzo [Dousson et al. 1999].

So, the representation in Kenzo, despite being the "most general" possible one, is quite efficient, since it is "minimal", in a certain sense, among all the isomorphic final objects in $\mathcal{C}^{cc}_{imp}$.

5.3 Families of simplicial sets in EAT

In the previous example, 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 as it is explained in the appendix. Now, we will show how the description given in [Rubio et al. 1997] for simplicial sets in the EAT system can be adapted to the hidden framework.

A *simplicial set* K is a graded set $\{K^n\}_{n\in\mathbb{N}}$, together with two family of maps $\delta^n_i\colon K^n \to K^{n-1}$, $n > 0$, $i = 1,\ldots,n$ and $\eta^n_i\colon K^n \to K^{n+1}$, $n \geq 0$, $i = 1,\ldots,n$, which satisfy known identities. An element of K^n is called *simplex of dimension* n and the maps δ and η are the *face* and *degeneracy* operators respectively. A simplex that is in the image of a degeneracy operator is called *abstract* simplex, otherwise it is called *geometric* simplex (see [May 1982] and [Lambán et al. 1999a] for the general definitions on simplicial sets).

The "minimal" signature to deal with simplicial sets is:

$$\begin{array}{rl} dgnr: & nat\ absm \to absm \\ face: & smst\ nat\ nat\ absm \to absm \\ eq^{absm}: & smst\ absm\ absm \to bool \\ eq^{gsm}: & smst\ gsm\ gsm \to bool \end{array}$$

where gsm denotes a set of *geometric simplices*, $absm$ the set of the *abstract simplices* and $smst$ is the *hidden* sort for simplicial sets. The operations $dgnr$ and $face$ represent, respectively, the degeneracy and face operators.

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

$$coer^{gsm}_{absm}\colon gsm \to absm$$

intended to transform a geometric simplex into the corresponding *non-degenerate* abstract simplex. This signature, together with the natural properties and partiality conditions, defines a category of algebras whose initial model is used to define D^{ss}_{absm}, the set of abstract simplices. A description for D^{ss}_{absm} is:

$$\{\langle (j_k,\ldots,j_1),a\rangle \mid \exists\ n\in\mathbb{N} \text{ such that } a\in B_n,\ k\in\mathbb{N},\ j_i\in\mathbb{N},\ \forall i=1,\ldots,k, \\ \text{and } 0\le j_1<\cdots<j_k\le n+k-1\}$$

(This description of the abstract simplices corresponds to the representation used in EAT, and is closer to the usual presentation used in simplicial topology; see [May 1982]. The description suggested by Kenzo is based on a very efficient numerical encoding of the *degeneracy list* $(j_k,\ldots,j_1)$, but we do not take into account the specification of these (implementation) technical matters due to efficiency of the programs.)

This completes the definition of a hidden signature for simplicial sets since the visible operations are fixed on the data domain. Again, partiality conditions on the operations are used to determine the graded structure on this data domain.

The 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 operations form a tuple $(f, =_{gsm})$ with $f\colon D^{ss}_{nat}\times D^{ss}_{nat}\times D^{ss}_{absm}\to D^{ss}_{absm}$ and $=_{gsm}\colon D^{ss}_{gsm}\times D^{ss}_{gms}\to D^{ss}_{bool}$ such that f is a representation of the face of a simplicial set and $=_{gsm}$ is an equality on the geometric simplices (see [Lambán et al. 1999a] for details).

As for chain complexes, we define $coer^{gsm}_{absm}(a) := \langle (),a\rangle$ for each geometric simplex a, and, if this operation is included in the signature, a polymorphic operation (which is present in Kenzo) appears:

$$face\colon smst\ nat\ nat\ gsm \to absm$$

This operation computes the faces of the geometric simplices and determines the face of the abstract simplices, so it can be used in the tuple of the final object instead of the faces on abstract simplices.

5.4 Families of simplicial sets in Kenzo

Whereas the above specification technique is suitable for the rest of the structures of EAT, it is not directly applied to the Kenzo structures due to their object orientation. In this system, structures are built taking advantage of the inheritance of CLOS by reusing of the common elements. Besides, we obtain, in an automatic way, some relevant polymorphic functions. This illustrates, in this particular case, the benefits of object-oriented programming from a software engineering point of view. For example, Kenzo implements simplicial sets as particular cases of a chain complexes, as simplicial sets can be interpreted as *Eilenberg-MacLane FD-complexes* (see [May 1982], page 93). Roughly speaking, a simplicial set $(X,\ face)$ is endowed with a differential structure

$$d_n \colon C_n(X) \to C_{n-1}(X)$$

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

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

The previous EAT system provides a construction function which builds this chain complex from a simplicial set, when needed (see [Rubio et al. 1997] and [Domínguez et al. 2001]).

Inheritance can seem strange in the case of the simplicial set/chain complex. It does not correspond to the `is_a` relationship, which is a “standard model” to understand object-oriented inheritance (see [Weber 1987], for instance). A typical example of inheritance between structures that follows this model is the case of the group/semigroup. In this case, it is said that a group is a semigroup. However, in Algebraic Topology it is usual to say that a simplicial set has (canonically) associated a chain complex, but it is not said that a simplicial set *is a* chain complex. Indeed, in this case, inheritance has been used to implement a `has_a` relationship (this shows how the notion of inheritance is difficult, conceptually, to apprehend; see [Breazu-Tannen et al. 1991], [Bruce 1992], [Taivalsaar 1996] or [Weber 1987] again). But, let us observe that, in category theory terminology, there is a functor F from the simplicial sets category to the chain complexes category. Thus, the situation is not so different from the group/semigroup case: there is a functor, even if it is not a *forgetful* functor. But, in such a context, given a functor $F \colon \mathcal{C} \to \mathcal{D}$ it is always possible to “enrich” the source category $\mathcal{C}$ to obtain a pair $[X, F(X)]$ for each object X of $\mathcal{C}$. Obviously, from this new category, there is a forgetful functor to $\mathcal{D}$ encoding the same construction F. This was the idea of Eilenberg-MacLane [May 1982], who replaced the notion of simplicial set by that of *FD-complex.* This was the approach chosen to write Kenzo and, coherently, it is the way of working reflected in our models of the following section.

To deal with this new situation, a first attempt is to use an *order-sorted specification* [Goguen and Meseguer 1992] which is the technique usually used in the hidden framework [Goguen and Malcolm 2000]: a new signature $\mathtt{SS}_{imp}$ is obtained by adding to the signature $\mathtt{CC}_{imp}$ of Section 5.2 the operations on simplicial sets of Section 5.3, and by declaring $smst < chcm$. However, this approach is not convenient because 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 [Goguen and Meseguer 1992]). But, the final objects in the previous sections and the appendix 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 [Mossakowski et al. 2000], in the context of the CoFI Algebraic Specification Language, CASL [CoFI Task Group 1999], 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 [Mossakowski et al. 2000], since it had previously been proposed by other authors in [Weber 1987, Bruce and Wegner 1990], 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 $\mathtt{SS}_{imp}$ signature by adding to $\mathtt{CC}_{imp}$ the operations on simplicial sets and a new operation:

$$coer^{smst}_{chcm} \colon smst \to chcm$$

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 complexes[1]), it allows to specify, at the syntactical level, all the Kenzo features (including polymorphism/overloading of operations) without including any redundant information. These coercion operations are present in other works. For example, they act implicitly in the algebraic structures in Coq [Geuvers et al. 2002], which simplifies the syntactic notation. However, we have preferred making explicit these operations to represent the inheritance relation between our structures.

This specification technique is generalized in the following section to model the inheritance derived from a forgetful relation between two structures in the Kenzo system.

6 Hidden specification of inheritance in Kenzo

Let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two categories and $F \colon \mathcal{T}^2 \to \mathcal{T}^1$ be a functor representing a forgetful relation between them. Let us assume that the modeling process explained in Section 4 can be accomplished for the $\mathcal{T}^1$ category. Basically, we have a signature $\Sigma^{1,eq}$, a data domain D^1 and a definition domain dom^1, and

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

we define $\mathcal{C}^1$ a full subcategory of $PAlg^{D^1,dom^1}(\Sigma^{1,eq})$ such that for each partial $\Sigma^{1,eq}$-algebra A of $\mathcal{C}^1$, the partial quotient algebra $A/_{eq^1_A}$ is an object of $\mathcal{T}^1$. Then, we can specify families of $\mathcal{T}^1$ objects if we construct a hidden signature $\Sigma^{1,eq}_{imp}$ and a subcategory $\mathcal{C}^1_{imp}$ of $HPAlg^{D^1,dom^1}(\Sigma^{1,eq}_{imp})$ such that an algebra A in $\mathcal{C}^1_{imp}$ verifies that for each element $a \in A_{imp_{\Sigma^{1,eq}}}$, the $\Sigma^{1,eq}$-algebra A^a is an object of $\mathcal{C}^1$.

In order to model the second category and the forgetful relation between them, we assume that we can define a signature $\Sigma^{2,eq}$, a data domain D^2 and a definition domain dom^2, such that $\Sigma^{1,eq} \subseteq \Sigma^{2,eq}$, $D^1 \subseteq D^2$ and $dom^1 \subseteq dom^2$. Then, we define a subcategory $\mathcal{C}^2$ of $PAlg^{D^2,dom^2}(\Sigma^{2,eq})$ such that for each partial $\Sigma^{2,eq}$-algebra A of $\mathcal{C}^2$, the partial quotient algebra $A/_{eq^2_A}$ is an object of $\mathcal{T}^2$. In addition, if we want to represent the functor F, it is necessary to impose that if we apply the functor F to the object $A/_{eq^2_A}$, we obtain the $\mathcal{T}^1$-object $(A|\Sigma^{1,eq})/_{eq^1_A}$. In this object, eq^1_A denotes the partial congruence in the restriction of the $\Sigma^{2,eq}$-algebra A to $\Sigma^{1,eq}$, $A|\Sigma^{1,eq}$, which is obtained when the equality operations of $\Sigma^{2,eq}$ are restricted to the operations derived from $\Sigma^{1,eq}$. In other words, we are assuming that the functor $F\colon \mathcal{T}^2 \to \mathcal{T}^1$ can be expressed in the model level through an operation $\tilde{F}\colon Obj(\mathcal{C}^2) \to Obj(\mathcal{C}^1)$ which builds the following commutative diagram among objects:

$$\begin{tikzcd}
Obj(\mathcal{C}^2) \arrow[r, "\tilde{F}"] \arrow[d, "/_{eq^2}"'] & Obj(\mathcal{C}^1) \arrow[d, "/_{eq^1}"] \\
\mathcal{T}^2 \arrow[r, "F"'] & \mathcal{T}^1
\end{tikzcd}$$

These conditions can seem quite demanding, but, as we will see in the examples, they appears naturally when we try to specify the forgetful relation between algebraic structures in general, and between the Kenzo structures in particular.

Then, we can specify families of $\mathcal{T}^2$ objects if we construct a hidden signature $\Sigma^{2,eq}_{imp}$. This signature contains the sorts and operations of $\Sigma^{1,eq}_{imp}$ together with the sorts $S^2 \setminus S^1 \cup \{imp_{\Sigma^{2,eq}}\}$ and the operations $\{imp\_\omega\colon imp_{\Sigma^{2,eq}} s_1 \dots s_n \to s\}_{\omega\colon s_1\dots s_n \to s \in \Omega^2 \setminus \Omega^1} \cup \{coer^{imp_{\Sigma^{2,eq}}}_{imp_{\Sigma^{1,eq}}} : imp_{\Sigma^{2,eq}} \to imp_{\Sigma^{1,eq}}\}$. In this signature, it is clear the syntactic reuse in the inheritance. Besides, we obtain polymorphism in some operations. The first signature operations can be applied to elements of the second hidden argument, in that case, the coercion operation acts.

Note that even if the signature $\Sigma^{2,eq}_{imp}$ has been completely defined, 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 $imp_{\Sigma^{2,eq}}$ to be a hidden sort. But, the nature of $imp_{\Sigma^{1,eq}}$ is more controversial.

If the structures of $\mathcal{T}^1$ and $\mathcal{T}^2$ are considered "at the same level", then $imp_{\Sigma^{2,eq}}$ and $imp_{\Sigma^{1,eq}}$ should be both hidden sorts. On the contrary, if $\mathcal{T}^1$ is considered a previous, auxiliary, data structure, then $imp_{\Sigma^{1,eq}}$ 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 $imp_{\Sigma^{2,eq}}$ and $imp_{\Sigma^{1,eq}}$ are declared hidden sorts, the signature $\Sigma^{2,eq}_{imp}$ can be directly considered a hidden signature, since the data domain part D^2 is fixed (sometimes using initial models as explained in examples of the previous section). Besides, the coercion operation is considered total, so, dom^2 determines the definition domain of this signature.

Now, to represent families of $\mathcal{T}^2$ objects a full subcategory $\mathcal{C}^2_{imp}$ of $HPAlg^{D^2,dom^2}(\Sigma^{2,eq}_{imp})$ is considered. An algebra A in this category verifies the following conditions. First, its restriction to $\Sigma^{1,eq}_{imp}$ is in $\mathcal{C}^1_{imp}$, i.e., $A|\Sigma^{1,eq}_{imp} \in \mathcal{C}^1_{imp}$. Second, for each element $a \in A_{imp_{\Sigma^{2,eq}}}$, the $\Sigma^{2,eq}$-algebra A^a, that is naturally defined when the element a is fixed as first component for the functions with sort $imp_{\Sigma^{2,eq}}$ and the element $coer^{imp_{\Sigma^{2,eq}}}_{imp_{\Sigma^{1,eq}}}(a)$ is fixed as first component for the functions with sort $imp_{\Sigma^{1,eq}}$, is an object of $\mathcal{C}^2$. The first condition represents the semantic reuse in the inheritance and the second allows polymorphism in some operations.

This situation is more complex than those analyzed in [Lambán et al. 1999b], as $\Sigma^{2,eq}$ is not a pure *deconstructor* signature (see [Lambán et al. 2003]). In other words,

$$coer^{imp_{\Sigma^{2,eq}}}_{imp_{\Sigma^{1,eq}}} : imp_{\Sigma^{2,eq}} \to imp_{\Sigma^{1,eq}}$$

is a *constructor* for the hidden sort $imp_{\Sigma^{1,eq}}$, and this implies that the techniques to be used are more complex. However, this kind of signatures are covered by the result on the existence of final objects in hidden categories of [Goguen and Malcolm 2000]. 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.

If we denote by $A^{1,can}$ the final object in the category $\mathcal{C}^1_{imp}$, we define a particular object in $\mathcal{C}^2_{imp}$, denoted by $A^{2,can}$ as follows. First, $A^{2,can}$ reuses $A^{1,can}$, i.e., $A^{2,can}|\Sigma^{1,eq}_{imp} = A^{1,can}$. Second, the elements of $A^{2,can}_{imp_{\Sigma^{2,eq}}}$ are tuples of partial functions: the functions of a tuple in $A^{1,can}_{imp_{\Sigma^{2,eq}}}$ that join with a function for each new operation in $\Sigma^{2,eq}$ (these specialized the former tuple) to define, together with D^2, an element of $\mathcal{C}^2$. Third, the coercion function acts over a tuple of $A^{2,can}_{imp_{\Sigma^{2,eq}}}$ forgetting the functions that correspond to operations of $\Omega^2 \setminus \Omega^1$. Finally, the rest of the functions are defined in a natural way. Then, $A^{2,can}$ is the final object of $\mathcal{C}^2_{imp}$.

Theorem 2. *The hidden category $\mathcal{C}^2_{imp}$ on the signature $\Sigma^{2,eq}_{imp}$, with two hidden sorts $imp_{\Sigma^{2,eq}}$ and $imp_{\Sigma^{1,eq}}$, has a final object.*

Proof. For each object $B \in \mathcal{C}^2_{imp}$, it is possible to define a (unique) weak hidden $\Sigma^{2,eq}_{imp}$-homomorphism, $h^{2,can}$, with total functions for the two hidden sorts defined as: $h^{2,can}_{imp_{\Sigma^{1,eq}}} = h^{1,can}_{imp_{\Sigma^{1,eq}}}$, where $h^{1,can}$ is the unique homomorphism between $B|\Sigma^{1,eq}_{imp}$ and $A^{1,can}$; and $h^{2,can}_{imp_{\Sigma^{2,eq}}} : B_{imp_{\Sigma^{2,eq}}} \to A^{2,can}_{imp_{\Sigma^{2,eq}}}$, such that $h^{2,can}_{imp_{\Sigma^{2,eq}}}(b) := (\omega_{B^b})_{\omega \in \Omega^{2,eq}}$ for each $b \in B_{imp_{\Sigma^{2,eq}}}$.

6.2 Hidden signature with a unique hidden sort

If we decide to declare $imp_{\Sigma^{1,eq}}$ as a *visible* sort, then things are easier, because the signature $\Sigma^{2,eq}_{imp}$ becomes a *deconstructor* signature and the general results for this particular kind of signatures of [Lambán et al. 1999b] and [Lambán et al. 2003] can be applied. But, in this case the data domain must be completed with a new set $D^2_{imp_{\Sigma^{1,eq}}}$.

The elements in $D^2_{imp_{\Sigma^{1,eq}}}$ should be interpreted as the ground on which the elements of $\mathcal{C}^2$ are built. Bearing in mind this interpretation, it is clear that a good candidate should be the carrier set on the sort $imp_{\Sigma^{1,eq}}$ in the final object obtained for $\mathcal{C}^1_{imp}$.

The fact that we usually define carrier sets for visible sorts through initial models (for example, combinations for chain complexes or abstract simplices for simplicial sets), while $D^2_{imp_{\Sigma^{1,eq}}}$ is fixed by means of a final model, reflects the different nature of these 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 (see [Rutten and Turi 1994], [Jacobs and Rutten 1997]).

Then, the next theorem follows from general results in [Goguen and Malcolm 2000] and [Lambán et al. 2003].

Theorem 3. *The hidden category $\mathcal{C}^2_{imp}$ on the signature $\Sigma^{2,eq}_{imp}$, with a unique hidden sort $imp_{\Sigma^{2,eq}}$, has a final object.*

This final object (as it is shown in [Lambán et al. 2003]) 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) $\Sigma^{2,eq}_{imp}$-algebras (obviously the final morphisms are different in both categories). However, the semantic interpretation of both situations has different connotations. If we have two hidden sorts, we can interpret it as a *reuse* relation. There are two structures at the same level and we reuse one to construct the other. If we have a unique hidden sort, we have a fixed structure previously built used as data domain in the other. This situation can be interpreted as an *use* relation. (A study on the differences between *use* and *reuse* relation can be found in [Meyer 1997].)

If we want to model the inheritance relationship implemented in Kenzo to deal with these structures, we should choose the *reuse* interpretation, which is close to the inheritance relationship, and not the *use* interpretation, which is closer to the *client* relationship. Nevertheless, as it is reported in [Meyer 1997], both interpretations can appear in inheritance relations.

7 Examples

In this section, we present two examples to illustrate the above technique. First, we explore the simple relation between semigroups and groups. In [Sergeraert 2001], the third author used similar examples to explain the implementation of the inheritance relation between structures in Kenzo. Second, we analyze the relationship between chain complexes and simplicial sets present in Kenzo.

7.1 Semigroups, groups

In this example, we work in a total frame without equalities (i.e., equalities are restricted to literal ones), focusing on the inheritance process.

A signature for semigroups is $\mathtt{SGRP}$, with one sort g and one operation $prd\colon g\ g \rightarrow g$, and a signature for groups $\mathtt{GRP}$ consists of $\mathtt{SGRP}$ enriched with two new operations $unt\colon \rightarrow g$, $inv\colon g \rightarrow g$. If a data domain $D = \{D_g\}$ is fixed, we can define a subcategory $\mathcal{C}^{SGRP}$ of $Alg^D(\mathtt{SGRP})$ whose algebras are semigroups on D, i.e., their operations verify the semigroup axiom $prd(a, prd(b, c)) = prd(prd(a, b), c)$. Similarly, we define the subcategory $\mathcal{C}^{GRP}$ of $Alg^D(\mathtt{GRP})$.

Now, if we apply the *imp* operation to $\mathtt{SGRP}$, we obtain a hidden signature $\mathtt{SGRP}_{imp}$ with g as visible sort and $imp_{\mathtt{SGRP}}$ as hidden sort, and an operation $imp\_prd\colon imp_{\mathtt{SGRP}}\ g\ g \rightarrow g$. Then, the subcategory $\mathcal{C}^{SGRP}_{imp}$ of $HAlg^D(\mathtt{SGRP}_{imp})$ whose algebras satisfy the axiom $imp\_prd(z, a, imp\_prd(z, b, c)) = imp\_prd(z, imp\_prd(z, a, b), c)$ gathers families of semigroups over D.

To specify families of groups over D, we can construct the signature $\mathtt{GRP}_{imp}$, which includes the sorts and operations of $\mathtt{SGRP}_{imp}$, a new sort $imp_{\mathtt{GRP}}$ and new operations $imp\_unt\colon imp_{\mathtt{GRP}} \rightarrow g$, $imp\_inv\colon imp_{\mathtt{GRP}}\ g \rightarrow g$, $coer^{imp_{\mathtt{GRP}}}_{imp_{\mathtt{SGRP}}}\colon imp_{\mathtt{GRP}} \rightarrow imp_{\mathtt{SGRP}}$. This signature is declared as a hidden signature with two hidden sorts $imp_{\mathtt{SGRP}}$, $imp_{\mathtt{GRP}}$. Then, we consider the subcategory $\mathcal{C}^{GRP}_{imp}$ of $HAlg^D(\mathtt{GRP}_{imp})$ whose algebras satisfy that their restriction to the $\mathtt{SGRP}_{imp}$-signature are objects of $\mathcal{C}^{SGRP}_{imp}$ and besides satisfy group axioms such as $imp\_prd(coer^{imp_{\mathtt{GRP}}}_{imp_{\mathtt{SGRP}}}(x), a, imp\_unt(x)) = a$, and so on.

The final object in the category $\mathcal{C}^{SGRP}_{imp}$ has as carrier set for the hidden sort tuples with one function. This function must represent the operation of a semigroup over D. The final object in the category $\mathcal{C}^{GRP}_{imp}$ has as carrier set for the hidden sort $imp_{\mathtt{GRP}}$ tuples of three functions: the one of the tuple for $\mathtt{SGRP}_{imp}$

which is specialized by two new functions to form a group over D. The coercion function acts as a forgetful function between both kind of tuples.

7.2 Chain complexes, simplicial sets.

At this point, we will use again the signatures $\mathtt{CC}_{imp}$, $\mathtt{SS}_{imp}$ and data domains and definition domains for them defined in Section 5. As is natural, we consider the carrier set for the generators of the chain complexes as the geometric simplices of the simplicial sets, i.e., $D^{cc}_{gnr} := D^{ss}_{gsm}$. Then, the final object for the simplicial sets admits the following *functional* description, denoted by B^{can}. The elements of B^{can}_{chcm} and B^{can}_{smst} are the pairs of functions $(d, =_{gnr})$ and $(f, =_{gsm})$ in the previous section. Then, the constructor is defined in a natural way: $(coer^{smst}_{chcm})_{B^{can}}(f, =_{gsm}) = (d_f, =_{gnr})$, where d_f is a representation of the differential FD-operator defined from f and $=_{gnr}$. This final object corresponds closely to the way in which simplicial sets have been implemented in the Kenzo system, as is illustrated in the appendix.

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

$$\begin{aligned} =_{gsm} &: D^{ss}_{gsm} \times D^{ss}_{gsm} \to D^{ss}_{bool} \\ =_{absm} &: D^{ss}_{absm} \times D^{ss}_{absm} \to D^{ss}_{bool} \\ f &: D^{ss}_{nat} \times D^{ss}_{nat} \times D^{ss}_{gsm} \to D^{ss}_{absm} \\ + &: D^{cc}_{cmbn} \times D^{cc}_{cmbn} \to D^{cc}_{cmbn} \\ d_f &: D^{ss}_{cmbn} \to D^{ss}_{cmbn} \end{aligned}$$

But the addition and the equality on abstract simplices are induced by the comparison test on geometric simplices $=_{gsm}$, whereas the differential is induced by this comparison test and the face operator f. Thus, in the final object, only $=_{gsm}$ and f are necessary. If a simplicial set is identified with the five operations above, it is clear that the coercion $coer^{smst}_{chcm}$ (and therefore the inheritance relationship) can be interpreted as a *forgetful* mapping.

8 Related work

In the previous sections we have mentioned, when necessary, references both to the theory of object-oriented programming and to the algebraic specification field. In this section, we will 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: [Lambán et al. 1999b] (where implementation issues were mathematically modeled), [Lambán et al. 2003] (where the relationship with hidden specifications and coalgebras was explored) and [Domínguez et al. 2001] (where our constructions were expressed in an institutional framework; for the concept of *institution*, see [Mosses 1989] or [Calmet and Tjandra 1993]). Our approach seems to be quite original, in the sense that it deals with the *modeling* of a system already produced: we do not try to explain the way in which a type system has been designed or used to implement a computer algebra program, but rather to obtain mathematical models (through algebraic specifications machinery) in order to have enough resources to *reason* on the *internal* processes of computation in the EAT and Kenzo systems.

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

In a first block of papers, we find those related to, or inspired by, the system Axiom (previously Scratchpad) [Jenks and Sutor 1992]. And the main reference should be the work explaining the ideas used in the development of Axiom, [Davenport and Trager 1990] for instance. There are several differences between our approach and the one of Axiom. A first source of differences stems from the systems themselves: Axiom has been designed to be a general purpose Computer Algebra system and Kenzo is a special purpose program created to compute homology and homotopy groups. In addition, the research in [Davenport and Trager 1990] focused on a system in progress aimed, among others things, at explaining the type system and using it to structure libraries; 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 to reason 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 [Loeckx et al. 1996] and [Goguen and Malcolm 2000], for instance, and not on higher-order techniques in algebraic specification, such as [Broy 1988] 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 and infinite data structures (as streams or series) can be managed in it (see [Burge and Watt 1987]), in EAT and Kenzo (and, indeed, in any *general* system of computation 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 [Rubio and Sergeraert 1993], [Rubio 2001]) must be considered together (see the notion of *effective homology* in the appendix): with the first ones, we can compute (the Betti numbers of a chain complex of *finite* type, 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 [Weber 1992, Weber 1994, Weber 1995] and Doye [Doye 1999, Doye 1997]. 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 with the aim of proving 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 are *a priori* known).

Last in this block, the Weyl computer algebra substrate [Zippel 1993] is a system written, as Kenzo, in CLOS. The Weyl system provides an *infrastructure to develop* computer algebra programs embedded in more general applications (integrating numerical methods and user interfaces, for example). However, our objective is to give a *superstructure to reason* 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 programs written on Weyl.

A second source of references on these topics comes from the works by Calmet et al. [Calmet and Tjandra 1993, Calmet et al. 1993, Homann and Calmet 1995]. These papers deal with the problem of *knowledge representation* and, particulary, with the representation of mathematical knowledge by means of algebraic structures. The formalism used for the specifications in the language FORMAL [Calmet and Tjandra 1993] is that of *unified algebras* [Mosses 1989], 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: computation with algebraic structures and computation with the *elements* of algebraic structures (these two layers are also explicitly present in Weyl [Zippel 1993], through the notions of *domain* and *domain element*, and implicitly in AXIOM [Jenks and Sutor 1992]). More work will be necessary to know whether the approaches of [Calmet and Tjandra 1993] and this paper can be formally integrated.

Finally, we must investigate whether our techniques can be applied to more general systems of Symbolic Computation such as Magma [Cannon and Playoust 1997], AXIOM [Jenks and Sutor 1992] or Aldor [ALDOR] (and, related to it, the application to the *Domains* package of Maple [Monagan et al. 2002]). Further investigation is needed at this point.

9 Conclusions and further work

This paper deals with the specification of the simple inheritance mechanisms used in a symbolic computation system, known as Kenzo. To this aim, two alternative approaches based on hidden specifications and coercions have been explored. The first approach implies declaring both sorts involved in inheritance as hidden sorts. In this case, the two structures which take part in the inheritance are considered at same level and the coercion operation is a constructor which obtains one structure after forgetting some characteristics of the other. The second idea involves declaring one of this sorts visible and then consider one structure as a previous auxiliary data structure to build the other structure. In our case, the first idea seems more natural, but implies more technical difficulties. These approaches model quite closely the simple inheritance mechanisms used in Kenzo and allow us to specify some polymorphic operations of this system. These techniques are used to specify some examples directly extracted from the system.

This work may continue along different lines. On the one hand, it will be necessary to translate these ideas 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. Another line is the study of the multiple inheritance present in Kenzo too. This is a difficult problem that requires further investigation. Finally, the questions raised in the previous section (in particular, the relationships with Weyl [Zippel 1993] and FORMAL [Calmet and Tjandra 1993]) must be elucidated.

Acknowledgements

This work has been partially supported by MEC, project MTM2006-06513 and by Comunidad Autónoma de La Rioja, project ANGI2005/19. We wish to thank to an anonymous referee for valuable comments which have improved the final version of this paper.

References

[Abadi and Cardelli 1996] M. Abadi and L. Cardelli, *A theory of objects.* Springer (1996).
[ALDOR] The Aldor programming language. `http://www.aldor.org`.
[Cannon and Playoust 1997] W. Bosma, J. Cannon, and C. Playoust, The Magma algebra system I: The user language. J. Symbolic Comput. 24 (1997) 235-265.

[Breazu-Tannen et al. 1991] V. Breazu-Tannen, T. Coquand, C. Gunter, and A. Scedrov, Inheritance as explicit coercion. Inform. and Comput. 93 (1991) 172-221.

[Broy 1988] M. Broy, Equational specification of partial higher-order algebras. Theoret. Comput. Sci. 57 (1988) 3-45.

[Bruce 1992] K. Bruce, The equivalence of two semantic definitions for inheritance in object-oriented languages. In *Mathematical Foundations of Programming Semantics*, edited by S.D. Brookes, M.G. Main, A. Melton, M.W. Mislove, and D.A. Schmidt. Lecture Notes in Comput. Sci. 598 (1992) 102-124.

[Bruce and Wegner 1990] K. Bruce and P. Wegner, An algebraic model of subtype and inheritance. In *Advances in Database Programming Language*, edited by F. Bancilhon and P. Buneman. ACM Press/Addison-Wesley (1990) 75-96.

[Burge and Watt 1987] W.H. Burge and S.M. Watt, Infinite structures in SCRATCHPAD II. In *Proc. of European Conference on Computer Algebra*, edited by J.H. Davenport. Lecture Notes in Comput. Sci. 378 (1987) 138-148.

[Calmet et al. 1993] J. Calmet, K. Homann, and I.A. Tjandra, Unified domains and abstract computational structures. In *Proc. of Artificial Intelligence and Symbolic Mathematical Computation*, edited by J. Calmet and J.A. Campbell. Lecture Notes in Comput. Sci. 737 (1993) 166-177.

[Calmet and Tjandra 1993] J. Calmet and I.A. Tjandra, A unified-algebra-based specification language for symbolic computing. In *Proc. of Design and Implementation of Symbolic Computation Systems*, edited by A. Miola. Lecture Notes in Comput. Sci. 722 (1993) 122-133.

[Cardelli 1988] L. Cardelli, A semantics of multiple inheritance. Inform. and Comput. 76(2) (1988) 138-164.

[Carlsson and Milgram 1995] G. Carlsson and R. J. Milgram, Stable homotopy and iterated loop spaces. In *Handbook of Algebraic Topology*, edited by I.M. James. North-Holland (1995) 505-583.

[Cerioli et al. 1999] M. Cerioli, T. Mossakowski, and H. Reichel, From total equational to partial first-order logic. In *Algebraic Foundations of Systems Specification*, edited by E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner. Springer (1999) 31-104.

[Davenport and Trager 1990] J.H. Davenport and B.M. Trager, Scratchpad's view of algebra I: Basic commutative algebra. In *Proc. of Design and Implementation of Symbolic Computation Systems*, edited by A. Miola. Lecture Notes in Comput. Sci. 429 (1990) 40-54.

[Domínguez et al. 2001] C. Domínguez, L. Lambán, V. Pascual, and J. Rubio, Hidden specification of a functional system. In *Proc. of Computer Aided Systems Theory*, edited by R. Moreno-Daz, B. Buchberger, and J.L. Freire. Lecture Notes in Comput. Sci. 2178 (2001) 555-569.

[Domínguez and Rubio 2001] C. Domínguez and J. Rubio, Modeling inheritance as coercion in a symbolic computation system. In *Proc. of International Symposium on Symbolic and Algebraic Computation*, edited by B. Mourrain. ACM Press (2001) 107-115.

[Domínguez et al. to appear] C. Domínguez, L. Lambán and J. Rubio, Object oriented institutions to specify symbolic computation systems. To appear in Rairo-Theor. Inform. Appl.

[Dousson 1999] X. Dousson, *Homologie effective des classifiants et calculs de groupes d'homotopie.* Thèse, Institut Fourier, Grenoble, France (1999).

[Dousson et al. 1999] X. Dousson, F. Sergeraert, and Y. Siret, *The Kenzo program.* Institut Fourier, Grenoble France (1999). `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo`.

[Doye 1997] N. Doye, *Order Sorted Computer Algebra and Coercions*, PhD thesis, University of Bath, United Kingdom (1997).

[Doye 1999] N. Doye, Automated coercion for AXIOM. In *Proc of International Symposium on Symbolic and Algebraic Computation*, edited by S. Dooley. ACM Press (1999) 229-235.

[Geuvers et al. 2002] H. Geuvers, R. Pollack, F. Wiedijk, and J. Zwanenburg, A constructive algebraic hierarchy in Coq. J. Symbolic Comput. 34 (2002) 271-286.

[Goguen and Malcolm 2000] J.A. Goguen and G. Malcolm, A hidden agenda. Theoret. Comput. Sci. 245(1) (2000) 55-101.

[Goguen and Meseguer 1992] J.A. Goguen and J. Meseguer, Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoret. Comput. Sci. 105(2) (1992) 217-273.

[Homann and Calmet 1995] K. Homann and J. Calmet, Combining theorem proving and symbolic mathematical computing. In *Integrating Symbolic Mathematical Computation and Artificial Intelligence*, edited by J. Calmet and J.A. Campbell. Lecture Notes in Comput. Sci. 958 (1995) 18-29.

[Jacobs and Rutten 1997] B. Jacobs and J. Rutten, A tutorial on (co)algebras and (co)induction. Bull. Eur. Assoc. Theoret. Comput. Sci. EATCS 62 (1997) 222-259.

[Jenks and Sutor 1992] R. D. Jenks and R. S. Sutor, *AXIOM: The Scientific Computation System.* Springer (1992).

[Kamir and Reddy 1994] S. Kamin and U. Reddy, Two semantic models of object-oriented languages. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*, edited by C. A. Gunter and J. C. Mitchell. MIT Press (1994) 463-495.

[Lambán et al. 2003] L. Lambán, V. Pascual, and J. Rubio, An object-oriented interpretation of the EAT system. Appl. Algebr. Eng. Commun. Comput. 14 (2003) 187-215.

[Lambán et al. 1999a] L. Lambán, V. Pascual, and J. Rubio, Simplicial sets in the EAT system. In *Proc. of Encuentros de Álgebra Computacional y Aplicaciones*, edited by I. Bermejo. Universidad de La Laguna, Spain (1999) 267-276.

[Lambán et al. 1999b] L. Lambán, V. Pascual, and J. Rubio, Specifying implementations. In *Proc. of International Symposium on Symbolic and Algebraic Computation*, edited by S. Dooley. ACM Press (1999) 245-251.

[Loeckx et al. 1996] J. Loeckx, H. D. Ehrich, and M. Wolf, *Specification of Abstract Data Types.* Wiley-Teubner (1996).

[Mac Lane 1994] S. Mac Lane, *Homology.* Springer, 4th edition, (1994).

[May 1982] J. P. May, *Simplicial Objects in Algebraic Topology.* Midway (1982).

[Meyer 1997] B. Meyer, *Object-Oriented Software Construction.* Prentice Hall, Second Edition (1997).

[Monagan et al. 2002] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S.M. Vorkoetter, J. McCarron, and P. DeMarco, *Maple 8 Introductory Programming Guide.* Waterloo Maple Inc. (2002).

[Mossakowski et al. 2000] T. Mossakowski, A. Haxthausen, and B. Krieg-Brückner, Subsorted partial higher-order logic as an extension of CASL. In *Recent Trends in Data Type Specification*, edited by D. Bert, C. Choppy, and P. Mosses. Lecture Notes in Comput. Sci. 1827 (2000) 126-145.

[Mosses 1989] P.D. Mosses, Unified algebras and institutions. In *Proc. of Logics in Computer Science*, IEEE Press (1989) 304-312.

[Rubio 2001] J. Rubio, Locally effective objects and Artificial Intelligence. In *Proc. of Artificial Intelligence and Symbolic Computation*, edited by J.A. Campbell and E. Roanes-Lozano. Lecture Notes in Artificial Intelligence 1930 (2001) 223-226.

[Rubio and Sergeraert 1993] J. Rubio and F. Sergeraert, Locally effective objects and algebraic topology. In *Computational Algebraic Geometry*, edited by F. Eyssette and A. Galligo. Progr. Math. 109 (1993) 235-251.

[Rubio et al. 1997] J. Rubio, F. Sergeraert, and Y. Siret, *EAT: Symbolic Software for Effective Homology Computation.* Institut Fourier, Grenoble, France (1997). `ftp://ftp-fourier.ujf-grenoble.fr/pub/EAT`.

[Rutten and Turi 1994] J.J.M.M. Rutten and D. Turi, Initial algebra and final coalgebra semantics for concurrency. In *A Decade of Concurrency, Reflections and Perspectives*, edited by J.W. de Bakker, W.P. de Roever, and G. Rozenberg. Lecture Notes in Comput. Sci. 803 (1994) 530-582.

[Sergeraert 1990] F. Sergeraert, Functional coding and effective homology. Astérisque 192 (1990) 57-67.

[Sergeraert 1994] F. Sergeraert, The computability problem in algebraic topology. Adv. Math. 104 (1994) 1-29.

[Sergeraert 2001] F. Sergeraert, Common lisp, typing and mathematics. Technical report, (2001). Satellite talk at the 2001 EACA Congress.

[Snyder 1987] A. Snyder, Inheritance and the development of encapsulated software components. In *Research Directions in Object-Oriented Programming*, edited by B. Shriver and P. Wegner. MIT Press (1987) 165-188.

[Taivalsaar 1996] A. Taivalsaari, On the notion of inheritance. ACM Computing Surveys 28(3) (1996) 438-479.

[CoFI Task Group 1999] The CoFI Task Group on Language Design, CASL, The Common Algebraic Specification Language - Summary. Version 1.0. Technical report (1999).

[Weber 1992] A. Weber, A type-coercion problem in computer algebra. In *Proc. Artificial Intelligence and Symbolic Mathematical Computation*, edited by J. Calmet, J.A. Campbell. Lecture Notes in Comput. Sci. 737 (1992) 188-194..

[Weber 1994] A. Weber, Algorithms for type inference with coercions. In *Proc. of International Symposium on Symbolic and Algebraic Computation*, edited by J. von zur Gathen and M. Giesbrecht. ACM Press (1994) 324-329.

[Weber 1995] A. Weber, On coherence in computer algebra. J. Symbolic Comput. 19 (1995) 25-38.

[Weber 1987] P. Wegner, The object-oriented classification paradigm. In *Research Directions in Object-Oriented Programming*, edited by B. Shriver and P. Wegner, MIT Press (1987) 479-560.

[Zippel 1993] R. Zippel, The Weyl computer algebra substrate. In *Proc. of Design and Implementation of Symbolic Computation Systems*, edited by A. Miola. Lecture Notes in Comput. Sci. 722 (1993) 303-318.

Appendix

The Kenzo program

The EAT and Kenzo systems are the first significant *machine programs* about classical Algebraic Topology. The data structures of these symbolic computation systems are rich enough to require an innovative analysis. This task is undertaken in this paper for some features of fragments of the Kenzo program. The purpose of this section is to briefly present the Kenzo program through some examples that reflect quite directly the implementation of some of its structures. For a systematic description of the Kenzo system, see [Dousson et al. 1999], and for a detailed study of the Kenzo characteristics, see [Dousson 1999].

For example, let us imagine that we want to calculate the homology group $H_5(\Omega^2 S^3)$, i.e. the fifth homology group of the second loop space of the 3-sphere S^3 (roughly speaking, a second loop space of some topological space X is the space of continuous maps from the 2-sphere S^2 to the space X). With Kenzo[2] we construct the 3-sphere,

```
>(setf s3 (sphere 3)) ✠
[K1 Simplicial-Set]
```

The program returns a simple external form of the assigned object, this is the Kenzo-object ♯1 (`K1`), a simplicial set, that is, a combinatorial version of the requested sphere, and this object is assigned to the symbol `s3`.

Then we construct the second loop space of this sphere,

```
> (setf l2s3 (loop-space s3 2)) ✠
[K18 Simplicial-Group]
```

The combinatorial version of the loop space is *highly* infinite: it is a combinatorial version of the space of *continuous* maps $S^2 \to S^3$ but functionally coded as a small set of functions in a simplicial group object, that is, a simplicial set endowed with a group structure compatible with the simplicial structure.

The homology groups of an object such as a chain complex, a simplicial set or a simplicial group, are computed by the Kenzo function `homology`. Now, we request the fifth homology group:

```
> (homology l2s3 5) ✠
homology in dimension 5:
Component Z/3Z
Component Z/2Z
---done---
```

and the result $H_5(\Omega^2 S^3) = \mathbb{Z}_2 \oplus \mathbb{Z}_3$ is obtained in 1 second with a 400 MHz PC.

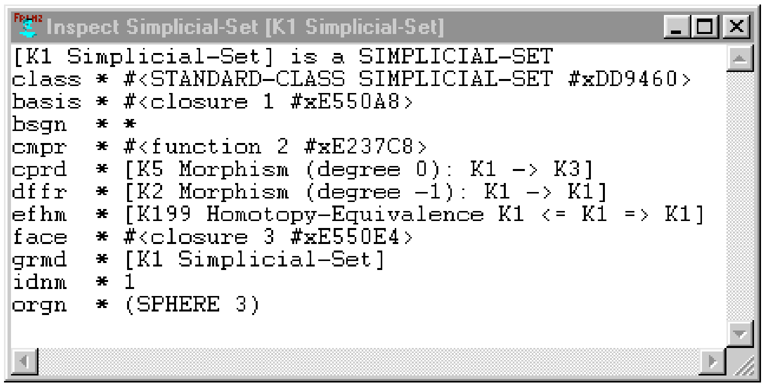
Let us study more carefully these structures examining, with the help of the CLOS function `inspect`, how they are stored on the computer memory.

```
> (inspect s3) ✠
```

The window displayed by this Lisp statement is shown in Figure 1.

Figure 1 is an example of how `inspect` displays a CLOS object: one line for each *slot* of the object; the name of the slot appears before the symbol `*`. Without going into excessive details, let us explain that, in our case, the essential slots of this *simplicial set* object are `basis` (encoding an algorithm that associates a list of simplices of dimension n to each natural number n), `cmpr` (encoding a comparison test between simplices) and `face` (encoding the face operators for each simplex). The rest of the slots appear by *inheritance* from other classes (see more details on inheritance below). Let us stress that the three most relevant slots are of *functional* nature (primitive functions or user-defined

[2] The small Lisp statements showed for illustration have really been run in the Kenzo program under Allegro Common Lisp. The Lisp prompt is here '>' and the maltese cross ✠ corresponds to the <Return> key that asks for the evaluation of the type-in statement.



```
[K1 Simplicial-Set] is a SIMPLICIAL-SET
class * #<STANDARD-CLASS SIMPLICIAL-SET #xDD9460>
basis * #<closure 1 #xE550A8>
bsgn  * *
cmpr  * #<function 2 #xE237C8>
cprd  * [K5 Morphism (degree 0): K1 -> K3]
dffr  * [K2 Morphism (degree -1): K1 -> K1]
efhm  * [K199 Homotopy-Equivalence K1 <= K1 => K1]
face  * #<closure 3 #xE550E4>
grmd  * [K1 Simplicial-Set]
idnm  * 1
orgn  * (SPHERE 3)
```

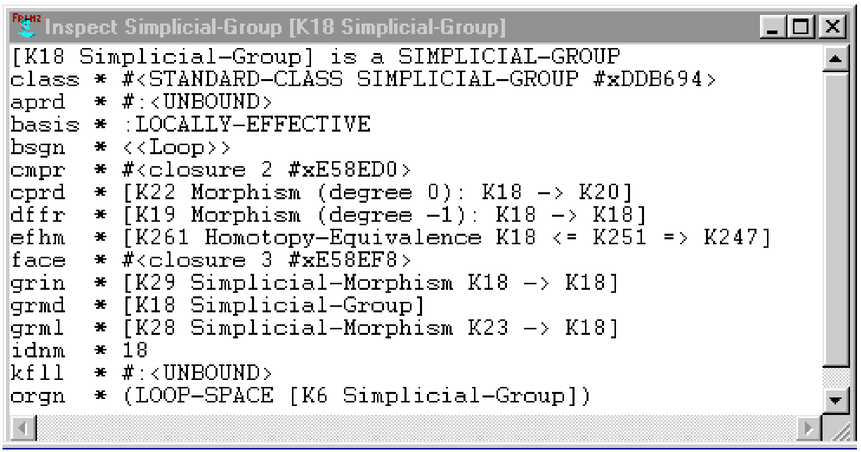
Figure 1: Inspect of `s3`

lexical closures). This feature has been very precisely reflected in the final object described in Section 6.

We display the `l2s3` structure as well so they may be compared.

> `(inspect l2s3)` ✠

See the inspect window in Figure 2.



```
[K18 Simplicial-Group] is a SIMPLICIAL-GROUP
class * #<STANDARD-CLASS SIMPLICIAL-GROUP #xDDB694>
aprd  * #:<UNBOUND>
basis * :LOCALLY-EFFECTIVE
bsgn  * <<Loop>>
cmpr  * #<closure 2 #xE58ED0>
cprd  * [K22 Morphism (degree 0): K18 -> K20]
dffr  * [K19 Morphism (degree -1): K18 -> K18]
efhm  * [K261 Homotopy-Equivalence K18 <= K251 => K247]
face  * #<closure 3 #xE58EF8>
grin  * [K29 Simplicial-Morphism K18 -> K18]
grmd  * [K18 Simplicial-Group]
grml  * [K28 Simplicial-Morphism K23 -> K18]
idnm  * 18
kfll  * #:<UNBOUND>
orgn  * (LOOP-SPACE [K6 Simplicial-Group])
```

Figure 2: Inspect of `l2s3`

Comparing Figure 1 and Figure 2, it is clear that `l2s3` contains all the slots of the previous simplicial set structure. This is due to the fact that in Kenzo the simplicial group class has been defined as a subclass of the simplicial set class.

In Category Theory, it is said that there is a *forgetful functor* from the category of simplicial groups to the category of simplicial sets. In our algebraic setting, this implies considering that the inheritance relationship *"forgets"* some of the slots. This characteristic of Kenzo has also been reflected in Section 6, on our algebraic specified models.

The new slots in this simplicial group class are `grml` and `grin`, which define the multiplication and inverse operations of the group, respectively. The `aprd` and `kfll` slots belong to this class because, besides, the simplicial group class inherits from another classes, particulary from the algebra class (`aprd` encodes the algebra product) and the Kan class (`kfll` encodes a Kan "hat"); see [Dousson et al. 1999] for a description of these structures.

An essential difference between the structures `s3` and `l2s3` is that the first one is a finite structure, *effective* in Sergeraert's terminology [Rubio and Sergeraert 1993], which is the direct implementation of a finite simplicial set; namely, a combinatorial version of the topological space S^3. Nevertheless, `l2s3` is an implementation of an infinite space, implementation that simulates this non-finiteness in such a way that all the information necessary for the calculations is accessible. This information will be always of a *local* nature, that is to say, related to one element or a finite set of elements which are near, in some (geometric or algebraic) sense. So, these structures are called *locally effective* objects. In this work, only locally effective objects have been considered. The `basis` slot reflects this characteristic, and stores a mapping from dimensions to lists of the non-degenerate simplices of the simplicial set if these lists are finite, i.e. corresponds with an effective object (this can be seen in Figure 1: the slot `basis` is a lexical closure). Otherwise (that is to say, if the simplicial set is infinite in some dimensions or if no information on its cardinality is available to the system), the keyword `:LOCALLY-EFFECTIVE` is stored in this slot (see Figure 2). Then, in the first case, we can ask for the list of non-degenerate simplices of `s3`, for instance in dimension 3:

```
  > (basis s3 3) ✠
  (S3)
```

It consists of a unique element, the "fundamental" simplex of S^3, which is called `s3` too. In the second case, we cannot obtain similar information from the `l2s3` object since the "list" of non-degenerate simplices in dimension 3 of the encoded space is infinite:

```
  > (basis l2s3 3) ✠
  ;; Error: The object [K18 Simplicial-Group] is
locally-effective.
```

Nevertheless, we can work with particular simplices of this simplicial group; we can compare if two simplices are equal, calculate their faces, etc. For example:

```
> (face l2s3 0 3 (loop3 1 (loop3 1 's3 1) 1)) ✠
<AbSm - <<Loop[<<Loop[0 s3]>>]>>>
```

which constructs the face ∂_0^3 of a simplex of `l2s3`.

Another important aspect is that operators have a homogeneous interface for every kind of object. So, to calculate the face ∂_0^3 of the "fundamental" simplex of `s3` we can use the same function:

```
> (face s3 0 3 's3) ✠
<AbSm 1-0 *>
```

In this operator, we can identify two different kinds of arguments: the first argument and the rest. The first argument determines the "ambient space" where the calculations will take place. As data objects, they are rather "hidden", because even if their structure can be explicitly explored by means of `inspect` (see Figures 1 and 2), their internal constitution cannot be shown: the slot values are lexical closures. The conclusion is that the objects associated to `s3` or `l2s3` only have *behavior*. They are *purely observational* (over given arguments for the lexical closures that compose them). By contrast, the rest of arguments are "visible", in the sense that their structure is completely known. This characteristic is also shared by the results of the applications, namely the corresponding abstract simplices, like `<AbSm 1-0 *>`: it is the degeneracy $\eta_1\eta_0$ of the base point (denoted by `*`) of the sphere; and their structure is completely "visible" from their machine representation. Note that "visible" data can be both constants (as the integers 0 or 3, for instance) or generated (as `<AbSm 1-0 *>`). This is to be compared with the standard approach in hidden specification, where each visible datum is considered as constant (see [Goguen and Malcolm 2000], page 63, and our presentation in Section 2). In Section 4 we present a more natural approach, modeling more accurately the situation in Kenzo (see in [Rubio 2001] how our technique can be also applied to other cases).

An important concept in Kenzo, which is stored in the slot `efhm`, is that of *effective homology* [Sergeraert 1994]. An object with *effective homology* is a "mixed" object with effective and locally effective features (an effective object such as `s3` can be considered, in particular, locally effective), that allows to benefit from both features: the locally effective coding of "infinite" objects allows to solve apparent difficulties of non-finiteness, while the effective coding allows to obtain information of global nature, such as the homology of the object. The two types of coding are related through hybrid objects, basically, homotopy equivalences between effective chain complexes and locally effective chain complexes. There is an example of such hybrid in the homotopy equivalence in the `efhm` slot (see Figure 2) of the object `l2s3` among the locally effective chain complexes `K18` and `K251` and the effective chain complex `K247`. Without going into details, this homotopy equivalence is a tool that allows to calculate the homology of a locally effective object, in this case `K18` (`l2s3`), by building an effective object

`K247` with the same homology (this object can be understood as a description of the homology of the previous object), through an intermediate object `K251` (see [Sergeraert 1994, Sergeraert 1990] for a detailed description). We can reach this auxiliary object:

```
> (K 251) ✠
[K251 Chain-Complex]
```

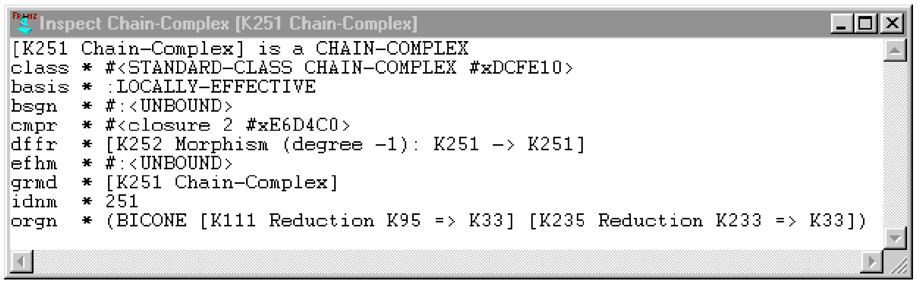
This object is really a chain complex. If we try to obtain the fifth homology group of this chain complex:

```
> (homology (K 251) 5) ✠
;; Error: I don't know how to determine the effective homology
of: [K251 Chain-Complex] (Origin: (BICONE [K111 Reduction
K95⇒K33] [K235 Reduction K233⇒K33]).
```

the program fails because the system has not enough knowledge to compute the homology of such a locally effective chain complex. Let us observe this object in detail:

```
> (inspect (K 251)) ✠
```

See the inspect window in Figure 3.



```
[K251 Chain-Complex] is a CHAIN-COMPLEX
class * #<STANDARD-CLASS CHAIN-COMPLEX #xDCFE10>
basis * :LOCALLY-EFFECTIVE
bsgn  * #:<UNBOUND>
cmpr  * #<closure 2 #xE6D4C0>
dffr  * [K252 Morphism (degree -1): K251 -> K251]
efhm  * #:<UNBOUND>
grmd  * [K251 Chain-Complex]
idnm  * 251
orgn  * (BICONE [K111 Reduction K95 => K33] [K235 Reduction K233 => K33])
```

Figure 3: Inspect of `K251`

The chain complex class is one of the most basic in Kenzo. Its essential slots (in the locally effective case) are `cmpr` (encoding the equality between elements) and `dffr` (encoding the differential maps). Both are functional, and this fact has also been reflected in the final object in the paper. The slots that have been not described correspond to the internal organization of the software (as `grmd`, `idnm` or `orgn`).

The chain complex class is inherited, in particular, by the simplicial set class (this is why every slot in Figure 3 also appears in Figures 1 and 2).

We insist that inheritance in the case of the simplicial group/simplicial set is quite natural because it is generally admitted that a simplicial group *is* (in particular) a simplicial set, and the `is_a` relationship is a "standard model" to understand object-oriented inheritance (see [Weber 1987], for instance). However, usually in Algebraic Topology it is said that a simplicial set has (canonically) associated a chain complex, but it is not said that a simplicial set *is a* chain complex. Indeed, in this case, inheritance has been used to implement a `has_a` relationship. But, note that, in category theory terminology, there is a functor F from the simplicial sets category to the chain complexes category and if we consider the idea of Eilenberg-MacLane [May 1982], who replaced the notion of simplicial set by that of *FD-complex*, we obtain a forgetful functor between this structure and the chain complex structure. This was the approach chosen to write Kenzo and, coherently, it is the way of working reflected in our models of Section 6.

At present the Kenzo program has already computed homology groups very difficult to obtain using old methods or unreachable so far with "classical" Algebraic Topology, even from a theoretical point of view. To illustrate this point, let us consider the homology group $H_5(\Omega^3\text{Moore}(\mathbb{Z}_2, 4))$[3] which is "in principle" reachable using old methods, see [Carlsson and Milgram 1995], but experience shows even the most skillful topologists have some difficulties to determine it. With the Kenzo program, we construct the third loop space of the Moore space:

```
> (setf l3m4 (loop-space (moore 2 4) 3)) ✠
[K291 Simplicial-Set]
```

and then its fifth homology-group is requested:

```
> (homology l3m4 5) ✠
homology in dimension 5:
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
Component Z/2Z
---done---
```

and the result $H_5(\Omega^3\text{Moore}(\mathbb{Z}_2, 4)) = \mathbb{Z}_2^5$ is obtained in 1m30s with a 400 MHz PC.

As the nature of this *mathematical* result (namely, that $H_5(\Omega^3\text{Moore}(\mathbb{Z}_2, 4))$ *is* $\mathbb{Z}_2^5$) can be a bit controversial (because no human seems capable of confirming or refuting this fact), it is quite clear that software reliability, in the case of the Kenzo program, becomes a main concern. We are trying to give the first steps in order to obtain a modeling of this system, with the aim of devising theoretical resources to reason on internal program processes.

[3] The space $\text{Moore}(\mathbb{Z}_2, 4)$ is a "canonical" connected space that only have non-trivial homology in dimension 4, namely $\mathbb{Z}_2$.