# A certified module of Simplicial Complexes for the Kenzo system*

Jónathan Heras and Vico Pascual and Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja,
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain).
{jonathan.heras, vico.pascual, julio.rubio}@unirioja.es

**Abstract.** Kenzo is a Computer Algebra System devoted to Algebraic Topology, and written in the Common Lisp programming language. In this paper, we give the programs which allow us to work with simplicial complexes in the Kenzo system, besides a complete automated proof of the correctness of our programs is provided. The proof is carried out using ACL2, a system for proving properties of programs written in (a subset of) Common Lisp.

## 1 Introduction

In the field of *Intelligent Information Processing*, mechanized reasoning systems provide a chance of increasing the reliability of software systems, namely *Computer Algebra Systems*. This paper is devoted to a concrete case of this topic.

The notion of simplicial complex, see [10], is the most elementary method to settle a connection between common "general" topology and homological algebra. The notion of topological space is too "abstract" in order to perform computations. A triangulation, by means of simplicial complexes, can be provided for "sensible" spaces, so every topological space can be considered as a simplicial complex, making the computations easier.

Nevertheless, many common constructions in topology are difficult to make explicit in the framework of simplicial complexes. It soon became clear in the forties the notion of simplicial set is much better. The reference [10] remains the basic reference in this subject.

The Kenzo system [5] is a Common Lisp program which works with the main mathematical structures used in Simplicial Algebraic Topology, namely it is able to work with simplicial sets. However the notion of simplicial complex is not included in the Kenzo system.

Kenzo was written mainly as a research tool and has got relevant results which have not been confirmed nor refuted by any other means. Then, the question of Kenzo reliability (beyond testing) arose in a natural way. Several works (see [2] and [9]) have focussed on studying the correctness of first order *fragments*

* Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01, and by European Community FP7, STREP project ForMath.

of Kenzo with the ACL2 theorem prover [8]. Other works have focussed on verifying the correctness of Kenzo algorithms using higher-order Theorem Provers tools such as Isabelle or Coq, see [3,4].

We have undertaken two tasks: on the one hand, the development of a new Kenzo module which integrates the notion of simplicial complex. On the other hand, certifying the correctness of this module using the ACL2 theorem prover.

The rest of this paper is organized as follows. Section 2 introduces the basic background about simplicial complexes and some algorithms about them. In Section 3 the new Kenzo module is presented; the ACL2 certification of that module is given in Section 4. Section 5 introduces our methodological approach to relate an efficient program with the proofs of properties in ACL2. This paper ends with a section of Conclusions and Further work.

We urge the interested reader to consult the complete development in [6].

## 2 Mathematical Preliminaries

In this section, we briefly provide the minimal mathematical background needed in the rest of the paper. We mainly focus on definitions. Many good textbooks are available for these definitions and results about them, the main one being maybe [10].

Let us start with the basic terminology. Let $V$ be an ordered set, called the *vertex set*. An *(ordered abstract) simplex* over $V$ is any ordered finite subset of $V$. An *(ordered abstract) $n$-simplex* over $V$ is a simplex over $V$ whose cardinality is equal to $n+1$. Given a simplex $\alpha$ over $V$, we call *faces* of $\alpha$ to all the subsets of $\alpha$.

**Definition 1** An *(ordered abstract) simplicial complex* over $V$ is a set of simplexes $\mathcal{K}$ over $V$ such that it is closed by taking faces (subsets); that is to say:

$$\forall \alpha \in \mathcal{K},\ if\ \beta \subseteq \alpha \Rightarrow \beta \in \mathcal{K}$$

Let $\mathcal{K}$ be a simplicial complex. Then the set $S_n(\mathcal{K})$ of $n$-simplexes of $\mathcal{K}$ is the set made of the simplexes of cardinality $n+1$ of $\mathcal{K}$.

**Example 2** Let us consider $V = (0,1,2,3,4,5,6)$.

The small simplicial complex drawn in Figure 1 is mathematically defined as the object:

$$\mathcal{K} = \left\{ \begin{array}{l} \emptyset, (0), (1), (2), (3), (4), (5), (6), \\ (0,1), (0,2), (0,3), (1,2), (1,3), (2,3), (3,4), (4,5), (4,6), (5,6), \\ (0,1,2), (4,5,6) \end{array} \right\}.$$

Note that, because the vertex set is ordered the list of vertices of a simplex is also ordered, which allows us to use a sequence notation $(\ldots)$ and not a subset notation $\{\ldots\}$ for a simplex and also for the vertex set $V$. It is also worth noting that simplicial complexes can be infinite. For instance if $V = \mathbb{N}$ and the simplicial complex $\mathcal{K}$ is $\{(n)\}_{n\in\mathbb{N}} \cup \{(n-1,n)\}_{n\geq 1}$, the simplicial complex obtained can be seen as an infinite bunch of segments.

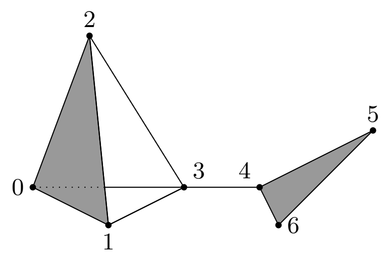

**Fig. 1.** Butterfly Simplicial Complex

**Definition 3** A *facet* of a simplicial complex $\mathcal{K}$ over $V$ is a maximal simplex with respect to the subset relation, $\subseteq$, among the simplexes of $\mathcal{K}$.

**Example 4** The facets of the small simplicial complex depicted in Figure 1 are:

$$\{(0,3),(1,3),(2,3),(3,4),(0,1,2),(4,5,6)\}$$

Let us note that a *finite* simplicial complex can be generated from its facets taking the set union of the power set of each one of their facets. In general, we have the following definition.

**Definition 5** Let $\mathcal{S}$ be a finite sequence of simplexes, then the set union of the power set of each one of the elements of $\mathcal{S}$ is, trivially, a simplicial complex called the *simplicial complex associated with* $\mathcal{S}$.

It is worth noting that the same simplicial complex can be generated from two different sequences of simplexes; in addition, the minimal sequence of simplexes which generates a finite simplicial complex is the sequence of its facets.

Then, the following algorithm can be defined.

**Algorithm 6**
*Input:* a sequence of simplexes $\mathcal{S}$.
*Output:* the associated simplicial complex with $\mathcal{S}$.

In spite of being a powerful tool, many common constructions in topology are difficult to make explicit in the framework of simplicial complexes. It soon became clear in the forties that the notion of simplicial set is much better.

**Definition 7** A *simplicial set* $K$, is a union $K = \bigcup_{q\geq 0} K^q$, where the $K^q$ are disjoints sets, together with functions:

$$\begin{array}{ll} \partial_i^q : K^q \to K^{q-1}, & q > 0, \quad i = 0,\dots,q, \\ \eta_i^q : K^q \to K^{q+1}, & q \geq 0, \quad i = 0,\dots,q, \end{array}$$

subject to the relations:

$$\begin{array}{lclcl}
(1)\ \partial_i^{q-1}\partial_j^q & = & \partial_{j-1}^{q-1}\partial_i^q & \text{if} & i<j, \\
(2)\ \eta_i^{q+1}\eta_j^q & = & \eta_j^{q+1}\eta_{i-1}^q & \text{if} & i>j, \\
(3)\ \partial_i^{q+1}\eta_j^q & = & \eta_{j-1}^{q-1}\partial_i^q & \text{if} & i<j, \\
(4)\ \partial_i^{q+1}\eta_i^q & = & \mathit{identity} & = & \partial_{i+1}^{q+1}\eta_i^q, \\
(5)\ \partial_i^{q+1}\eta_j^q & = & \eta_j^{q-1}\partial_{i-1}^q & \text{if} & i>j+1.
\end{array}$$

The $\partial_i^q$ and $\eta_i^q$ are called *face* and *degeneracy* operators respectively.

The elements of $K^q$ are called *q-simplexes*. A simplex $x$ is called *degenerate* if $x=\eta_i y$ for some simplex $y$ and some degeneracy operator $\eta_i$; otherwise $x$ is called *non degenerate*.

There exists a link between the notion of simplicial set and the one of simplicial complexes.

**Definition 8** Let $\mathcal{SC}$ be an (ordered abstract) simplicial complex over $V$. Then the *simplicial set* $K(\mathcal{SC})$ *canonically associated* with $\mathcal{SC}$ is defined as follows. The set $K^n(\mathcal{SC})$ is $S_n(\mathcal{SC})$, that is, the set made of the simplexes of cardinality $n+1$ of $\mathcal{SC}$. In addition, let $(v_0,\ldots,v_q)$ be a $q$-simplex, then the *face* and *degeneracy* operators of the simplicial set $K(\mathcal{SC})$ are defined as follows:

$$\begin{array}{l}
\partial_i^q((v_0,\ldots,v_i,\ldots,v_q)) = (v_0,\ldots,v_{i-1},v_{i+1},\ldots,v_q), \\
\eta_i^q((v_0,\ldots,v_i,\ldots,v_q)) = \quad (v_0,\ldots,v_i,v_i,\ldots,v_q).
\end{array}$$

That is, the face operator $\partial_i^q$ removes the vertex in the position $i$ of a $q$-simplex, and the degeneracy operator $\eta_i^q$ duplicates the vertex in the position $i$ of a $q$-simplex.

Then, the above definition provides us the following algorithm.

**Algorithm 9**
*Input:* a simplicial complex $\mathcal{SC}$.
*Output:* the simplicial set $K(\mathcal{SC})$ canonically associated with $\mathcal{SC}$.

## 3 A new Kenzo module

In the current www-available version of Kenzo, the notion of simplicial complex is not included. Then, we have developed a new Common Lisp module to enhance the Kenzo system with this notion. Our programs implement algorithms 6 and 9. The following lines are devoted to explain the essential part of these programs.

First of all, let us note that the vertex set $V$ in our programs is $\mathbb{N}$; besides, we represent an $n$-simplex as a strictly ordered list of $n+1$ natural numbers that represent the vertices of the simplex. For instance, the 2-simplex with vertices 0, 1 and 3 is represented as the list `(0 1 3)`. Moreover, a simplicial complex is represented, in our system, by means of a list of simplexes.

The first of our programs implements Algorithm 6, that is, the functions which generate a simplicial complex from a sequence of simplexes. The description of the main function in charge of this task is shown here:

**simplicial-complex-generator ls** *[Function]*

From a list of simplexes, `ls`, this function generates the associated simplicial complex. The implementation of this function is split in two steps. The former one, consists of gathering the lists of simplexes coming from the computation of the simplicial complex of each simplex of `ls`; as a result a list of simplexes is produced but probably with duplicate elements. The second step is devoted to remove the duplicate elements.

The second program implements Algorithm 9. It generates the simplicial set canonically associated with a simplicial complex as a `Simplicial-Set` Kenzo class instance. The main function is:

**ss-from-sc *simplicial-complex*** *[Function]*

Build an instance of the `Simplicial-Set` Kenzo class which represents the simplicial set canonically associated with a simplicial complex, *simplicial-complex*, see Definition 8, using some auxiliar functions which are necessary to define simplicial sets in Kenzo.

To provide a better understanding of the new tools, an elementary example of their use is presented now. Let us consider the list of facets of the butterfly simplicial complex presented in Example 4. From these facets, we can construct the butterfly simplicial complex with our program:

```
> (setf butterfly (simplicial-complex-generator
                    '((0 3) (1 3) (2 3) (3 4) (0 1 2) (4 5 6))))
((0 3) (0) (3) (1 3) (1) (2 3) (2) (3 4) (4) (0 1 2) (0 1) ...)
```

Once we have constructed this simplicial complex, we can build the simplicial set canonically associated with the butterfly simplicial complex by means of the instruction:

```
> (setf butterfly-ss (ss-from-sc butterfly)) ✠
[K1 Simplicial-Set]
```

Finally, we can determine its homology groups thanks to the Kenzo kernel.

```
> (homology butterfly-ss 0 2) ✠
Homology in dimension 0:
Component Z
Homology in dimension 1:
Component Z
Component Z
```

To be interpreted as stating $H_0(butterfly) = \mathbb{Z}$, $H_1(butterfly) = \mathbb{Z} \oplus \mathbb{Z}$.

## 4 Certification of the Kenzo module

As we have just said, we want to formalize in ACL2 the correctness of both the `simplicial-complex-generator` and `ss-from-sc` functions; that is to say, our implementation of algorithms 6 and 9. Since both Kenzo and ACL2 are Common Lisp programs we can verify the correctness of these functions in ACL2.

The formalization of `simplicial-complex-generator` in ACL2 is split in two steps. First of all, we need some auxiliary functions which define the necessary concepts to prove our theorems. Namely, we need to define the notions of *simplex*, *list of simplexes*, *set of simplexes*, *face* and *member* in ACL2. Subsequently, lemmas stating the correctness and completeness of our implementation of Algorithm 6 are proved; for instance the following one.

**ACL2 Lemma 10** Let $ls$ be a list of simplexes, then (`simplicial-complex-generator` $ls$) builds a set of simplexes.

```
(defthm simplicial-complex-generator-constructs-simplicial-complex-1
  (implies (list-of-simplexes-p ls)
           (set-of-simplexes-p (simplicial-complex-generator ls))))
```

The proof of lemmas like the above one, in spite of involving some auxiliary result, can be proved without any special hindrance due to the fact that our programs follows simple inductive schemas that are suitable for the ACL2 heuristics. Eventually, we can prove the following theorem.

**ACL2 Theorem 11** Let $ls$ be a list of simplexes, then (`simplicial-complex-generator` $ls$) constructs the simplicial complex associated with $ls$.

Finally, the certification of the correctness of `ss-from-sc`, that constructs from a simplicial complex the simplicial set canonically associated, is provided by means of the Generic Simplicial Set theory tool [7]. This tool reduces the proving effort for each family of simplicial sets, letting ACL2 automates the routine parts of the proof. In this way, the fact that from a simplicial complex the program *really* constructs a simplicial set is proven.

The task of certifying the correctness of our implementation of Algorithm 9, that is to say, the `ss-from-sc` function, has not been undertaken from scratch, but we have used a previous work presented in [7] that allows us to prove the correctness of simplicial sets constructed in the Kenzo object. Namely, if a Kenzo object fulfills some minimal conditions, we have developed an ACL2 theory which asserts that the Kenzo object is a simplicial set.

In this way, the proof effort is considerably reduced to prove the correctness of `ss-from-sc` since we only need to prove 4 (easy) properties, and the tool presented in [7] automatically generates the proof of the correctness of our implementation. Then, we have the following theorem.

**ACL2 Theorem 12** Let $sc$ be a simplicial complex, then (`ss-from-sc` $sc$) constructs a simplicial set.

## 5 Two equivalent programs

The implementation of the `simplicial-complex-generator` function is suitable for the induction heuristics of the ACL2 theorem prover. However, it is an inefficient design, so, it can produce undesirable situations. For instance, if we try to build a simplicial complex from a list of 11613 simplexes, an error message will be shown:

```
> (simplicial-complex-generator ...) ✠
Error: Stack overflow (signal 1000)
[condition type: SYNCHRONOUS-OPERATING-SYSTEM-SIGNAL]
```

This kind of error occurs when too much memory is used on the data structure that stores information about the active computer program.

In order to overcome this drawback, an efficient program has been implemented, this new function is called `optimized-simplicial-complex-generator`. This new program relies on the *memoization* technique, a well known artifact of Artificial Intelligence. Memoization is used primarily to speed up computer programs. A memoized function "remembers" the results corresponding to some set of specific inputs. Subsequent calls with remembered inputs return the remembered result rather than recalculating it.

However, no reward comes without a corresponding price and, the optimized program can not be implemented in ACL2 (since ACL2 is an applicative subset of Common Lisp). In order to deal with this pitfall we have based on the work presented in [1], where the authors coped with a similar problem, but related to already implemented Kenzo code fragments.

Let us enumerate the characteristics of our situation:

– `simplicial-complex-generator` program is
  - specially designed to be proved;
  - programmed in ACL2 (and, of course, Common Lisp);
  - not efficient;
  - tested;
  - proved in ACL2.

– `optimized-simplicial-complex-generator` program is
  - specially designed to be efficient;
  - written in Common Lisp;
  - efficient;
  - tested;
  - unproved.

In our approach, `simplicial-complex-generator` is *supposed to be equivalent* to `optimized-simplicial-complex-generator`. But we do not pretend to prove this equivalence: this option would lead us to a form of ill-founded recursion. Our aim should be to use the *highly reliable* `simplicial-complex-generator` to perform automated testing of the *efficient* `optimized-simplicial-complex-generator`.

The following toy program will illustrate this idea:

```
(defun automated-testing ()
  (let ((cases (generate-test-cases 100000)))
    (dolist (case cases)
      (if (not (equal-as-sc (simplicial-complex-generator case)
                    (optimized-simplicial-complex-generator case)))
          (report-on-failure case)))))
```

With this intensive testing, it is hoped that `simplicial-complex-generator` accurately models `optimized-simplicial-complex-generator`, and then our strategy could be safely applied.

## 6 Conclusions and further work

The module presented in this paper allows one to work with simplicial complexes, a sensible representation for topological spaces. The implementation has been written in Common Lisp, enhancing the Kenzo system but also allowing us to certify the correctness of the programs in ACL2.

The development of new certified Kenzo modules (for instance, a module to work with digital images) and the verification of already implemented Kenzo programs (for instance, the computation of homology groups) remains as further work.

## References

1. M. Andrés, L. Lambán, and J. Rubio. Executing in Common Lisp, Proving in ACL2. *Lectures Notes in Computer Science*, 4573:1–12, 2007.
2. M. Andrés, L. Lambán, J. Rubio, and J. L. Ruiz-Reina. Formalizing Simplicial Topology in ACL2. *Proceedings of ACL2 Workshop 2007*, pages 34–39, 2007.
3. J. Aransay, C. Ballarin, and J. Rubio. A mechanized proof of the Basic Perturbation Lemma. *Journal of Automated Reasoning*, 40(4):271–292, 2008.
4. C. Domínguez and J. Rubio. Effective Homology of Bicomplexes, formalized in Coq. *Theoretical Computer Science*, 412:962–970, 2011.
5. X. Dousson, J. Rubio, F. Sergeraert, and Y. Siret. The Kenzo program. Institut Fourier, Grenoble, 1998. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
6. J. Heras. ACL2 verification of Simplicial Complexes programs for the Kenzo system. University of La Rioja, 2010. `http://www.unirioja.es/cu/joheras/simplicial-complexes.html`.
7. J. Heras, V. Pascual, and J. Rubio. Proving with ACL2 the correctness of simplicial sets in the Kenzo system. In *Proceedings of 20th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'2010)*, volume 6564 of *Lectures Notes in Computer Science*, pages 37–51. Springer-Verlag, 2011.
8. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An approach.* Kluwer Academic Publishers, 2000.
9. F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. ACL2 verification of simplicial degeneracy programs in the Kenzo system. *Lectures Notes in Computer Science*, 5625:106–121, 2009.
10. J. P. May. *Simplicial objects in Algebraic Topology*, volume 11 of *Van Nostrand Mathematical Studies.* 1967.