Certified Symbolic Manipulation: Bivariate Simplicial Polynomials

Laureano Lambán

Dept. of Mathematics and Computation
University of La Rioja
Calle Luis de Ulloa s/n.
26004 Logroño, Spain
lalamban@unirioja.es

Francisco J. Martín–Mateos

Dept. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes, s/n.
41012 Sevilla, Spain
fjesus@us.es

Julio Rubio

Dept. of Mathematics and Computation
University of La Rioja
Calle Luis de Ulloa s/n.
26004 Logroño, Spain
julio.rubio@unirioja.es

José–Luis Ruiz–Reina

Dept. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes, s/n.
41012 Sevilla, Spain
jruiz@us.es

ABSTRACT

Certified symbolic manipulation is an emerging new field where programs are accompanied by certificates that, suitably interpreted, ensure the correctness of the algorithms. In this paper, we focus on algebraic algorithms implemented in the proof assistant ACL2, which allows us to verify correctness in the same programming environment. The case study is that of *bivariate simplicial polynomials*, a data structure used to help the proof of properties in Simplicial Topology. Simplicial polynomials can be computationally interpreted in two ways. As symbolic expressions, they can be handled algorithmically, increasing the automation in ACL2 proofs. As representations of functional operators, they help proving properties of categorical morphisms. As an application of this second view, we present the definition in ACL2 of some morphisms involved in the Eilenberg-Zilber reduction, a central part of the *Kenzo* computer algebra system. We have proved the ACL2 implementations are correct and tested that they get the same results as *Kenzo* does.

*Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842, and by European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

1. INTRODUCTION

A *certificate* is some datum added to an expression that can be used to verify (in an independent way) some property of the expression. The typical example is the certificate used to check the non-primality of a (big) natural number n: a pair of numbers whose product is equal to n. Many contributions have been made in this area, in particular in the area of *algebraic certificates* (see, for instance, [3]).

In an extreme case, we could identify a certificate with a complete proof, encoded in a concrete formalism, of the property under study. This is the point of view in the *proof-carrying code* paradigm [11], where a code *producer* attaches a proof of some property to the program; then a code *consumer* invokes a proof checker, that ensures a safety condition.

In this paper, we explore another path: programs, proofs, certificates and checking are generated in the same environment, the one of the theorem prover ACL2 [6]. In ACL2, once a proof has been mechanized, a certificate file is created, and the system only checks this file, without redoing the proof (which is, usually, a very time-consuming task) every time the mechanization is included in further developments.

Our area of application is *verified computer algebra*, and specifically algebraic computing in Algebraic Topology. Even more concretely, we are interested in applying formal

methods to specify and analyze the behavior of *Kenzo* [5], a program to compute in Homological Algebra and Simplicial Topology. *Kenzo* is a Common Lisp program, created by F. Sergeraert, that can deal with infinite dimensional spaces, and is able to compute results that cannot be determined by any other means (theoretical or computational). In [13] a theorem corrected thanks to *Kenzo* is presented, together with other results computed with *Kenzo* which seem out of reach for any other method.

Due to these *Kenzo* features, a project was launched some years ago to formally study its correctness, trying to give to *Kenzo* results an status as close as possible to standard mathematical properties. To this aim, different methods and tools have been used. For instance, the proof assistants Isabelle and Coq have been used to model important algorithms appearing in *Kenzo* [2, 4].

Both Isabelle/HOL and Coq are very powerful tools (in particular, both are based on higher-order logic), but they are far from the *Kenzo* programming language: Common Lisp. So it is natural to use ACL2, a theorem prover intimately linked to this language. Even if ACL2 is not suitable to model all *Kenzo* features (*Kenzo* uses higher-order functional programming, while ACL2 is a first-order tool; this explains the role of Isabelle/HOL and Coq in the global project), it is superior to any other tool to formalize the *actual Kenzo* source code (for an example of such a formalization, see [9]).

In this area of ACL2 applications to Algebraic and Simplicial Topology, several contributions have been already made [1, 7, 9]. Our last development is a complete ACL2 proof of the so-called Eilenberg-Zilber theorem [8]. This is a central theorem in Computational Algebraic Topology. This fundamental aspect of Eilenberg-Zilber is reflected in its computational counterpart: experimental studies of *Kenzo* log files showed that most of the running time is devoted to Eilenberg-Zilber computations (and more concretely to the computation of the morphism which will be called *Shih* later). Thus, giving a mechanized proof of it seems a good challenge to demonstrate the usability of this kind of formal methods in computer algebra verification.

The formal proof of the Eilenberg-Zilber Theorem needed around 13000 lines of ACL2 code (326 definitions and 1368 lemmas and theorems whitout the generic development of polynomials and basic arithmetic properties), so being quite a big endeavour. In this paper, instead of explaining the main lines of the proof, we focus on some technical achivements, that we consider can be more interesting for a wider audience in symbolic computation. Specifically, we introduce the notion of *bivariate simplicial polynomial*, a data structure instrumental to getting a greater automation in the mechanized proofs. Let us remark that, when choosing the data structure representations, we have followed the most natural ideas (emulating, to a certain extent, the Kenzo way of working), guessing that they will produce the easiest proofs. Our data structures are inspired by those of Kenzo, but adapting them to the constraints imposed by ACL2. For instance, when in Kenzo an array is chosen to represent an entity, our ACL2 version would be a list.

In addition, simplicial polynomials can be executed, giving an experimental flavour to the deductive standard machinery, and can be evaluated (on concrete topological spaces), allowing an *automated testing* for some parts of the *Kenzo* system. To illustrate the way of working in ACL2, we chose an operation on simplicial polynomials called *derivative*, essential to define and prove properties about the abovementioned *Shih* morphism.

The organization of the paper is as follows. Next section is devoted to state the mathematical problem, while Section 3 deals with the description of the ring of bivariate simplicial polynomials. We explain the notion of derivative in Section 4. Experimental results and computational aspects are presented in Section 5. The paper ends with conclusions and the bibliography.

2. MATHEMATICAL CONCEPTS AND APPLICATIONS

In this section, we introduce briefly the mathematical notions needed to understand the rest of the paper (for details and context about Simplicial Topology, see [10]).

DEFINITION 1. *A* simplicial set K *is a graded set* $\{K_n\}_{n\in\mathbb{N}}$ *together with functions:*

$$\partial_i^n : K_n \to K_{n-1}, \quad n > 0, \quad i = 0, \ldots, n,$$
$$\eta_i^n : K_n \to K_{n+1}, \quad n \geq 0, \quad i = 0, \ldots, n,$$

subject to the following equations:

$$\begin{array}{lrcll}
(1) & \partial_i^{n-1}\partial_j^n & = & \partial_j^{n-1}\partial_{i+1}^n & \text{if} \quad i \geq j, \\
(2) & \eta_i^{n+1}\eta_j^n & = & \eta_{j+1}^{n+1}\eta_i^n & \text{if} \quad i \leq j, \\
(3) & \partial_i^{n+1}\eta_j^n & = & \eta_{j-1}^{n-1}\partial_i^n & \text{if} \quad i < j, \\
(4) & \partial_i^{n+1}\eta_j^n & = & \eta_j^{n-1}\partial_{i-1}^n & \text{if} \quad i > j+1, \\
(5) & \partial_i^{n+1}\eta_i^n & = & \partial_{i+1}^{n+1}\eta_i^n & = \quad id^n,
\end{array}$$

The elements of K_n are called simplices of *dimension* n, or simply *n-simplices*. The functions ∂ and η are called face and degeneracy operators, respectively. A simplex x is called *degenerate* if it can be written as $x = \eta_i y$ for some index i and some simplex y[1]. Otherwise, it is called *non-degenerate*. The set of non-degenerate n-simplices of K is denoted by K_n^{ND}.

With any simplicial set K we can associate an algebraic structure $C(K)$, a chain complex, in such a way that the homology of K is exactly the homology of $C(K)$:

DEFINITION 2. *A* chain complex *is a family of pairs* $C := \{(C_n, d_n)\}_{n\in\mathbb{Z}}$ *where each* C_n *is an abelian group, and each* d_n *is a homomorphism from* C_n *to* C_{n-1} *such that the boundary condition holds:* $d_n \circ d_{n+1} = 0$.

[Given a chain complex C, the boundary condition implies $\text{Im } d_{n+1} \subseteq \text{Ker } d_n$; then the *homology groups* of C are well-defined: $H_n(C) = \text{Ker } d_n / \text{Im } d_{n+1}$. These homology groups are the objects *Kenzo* finally computes.]

Let K be a simplicial set. For each $n \in \mathbb{N}$, let us consider $\mathbb{Z}[K_n^{ND}]$, the free abelian group generated by the non-degenerate n-simplices, denoted as $C_n(K)$. Then, the elements of such a group are formal linear combinations $\sum_{j=1}^r \lambda_j x_j$, where $\lambda_j \in \mathbb{Z}$ and $x_j \in K_n^{ND}, \forall j = 1, \ldots, r$. These linear combinations are called *chains of simplices* or, in short, *chains*.

Now, given $n > 0$, we introduce the homomorphism $d_n : C_n(K) \to C_{n-1}(K)$, first defining it over each generator, and then extending it by linearity. Given $x \in K_n^{ND}$, define $d_n(x) = \sum_{i=0}^n (-1)^i \partial_i(x)$, where a term $\partial_i(x)$ is erased

[1]Note that, if the context is clear enough, the superindexes denoting dimension will be skipped.

when it is degenerate. It can be proved that the equations in the definition of simplicial set imply that $d_n \circ d_{n+1} = 0$, $\forall n \in \mathbb{N}$. That is to say, the family $\{d_n\}_{n\in\mathbb{N}}$ defines a *differential* (or boundary) homomorphism on the graded group $\{C_n(K)\}_{n\in\mathbb{N}}$, and then, the family of pairs $\{(C_n(K), d_n)\}_{n\in\mathbb{N}}$ is the *chain complex*[2] associated to the simplicial set K, denoted by $C(K)$.

An alternative definition can be given, by taking as generators *all* the simplices (degenerate and non-degenerate ones) in each dimension, and with the same expression for differentials: $\sum_{i=0}^{n}(-1)^i\partial_i(x)$ (now, it is not necessary to drop out any term). This (bigger) chain complex associated with a simplicial set K has homology groups canonically isomorphic to those of $C(K)$; more concretely, the so-called Normalization Theorem establishes a *reduction* between both chain complexes associated with a simplicial set (that reduction was programmed in ACL2 and proved correct in [7]).

Definition 3. *Given two chain complexes $C^1 := \{(C_n^1, d_n^1)\}_{n\in\mathbb{Z}}$ and $C^2 := \{(C_n^2, d_n^2)\}_{n\in\mathbb{Z}}$, a* reduction *between them is a triple (f, g, h) where $f : C^1 \to C^2$ and $g : C^2 \to C^1$ are chain morphisms (that is to say, they are families of homomorphisms $f_n : C_n^1 \to C_n^2$ and $g_n : C_n^2 \to C_n^1$ such that $f_{n-1} \circ d_n^1 = d_n^2 \circ f_n$ and $g_{n-1} \circ d_n^2 = d_n^1 \circ g_n$), and h is a family of homomorphisms (called* homotopy operator*) $h_n : C_n^1 \to C_{n+1}^1$ satisfying*

1. $f_n \circ g_n = id$,
2. $d_{n+1}^1 \circ h_n + h_{n-1} \circ d_n^1 + g_n \circ f_n = id$,
3. $f_{n+1} \circ h_n = 0$,
4. $h_n \circ g_n = 0$, *and*
5. $h_{n+1} \circ h_n = 0$.

We denote a reduction as $(f, g, h) : C^1 \Longrightarrow C^2$. The main property of a reduction is that it establishes a canonical isomorphism between the homology groups of C^1 and C^2. In fact, the components f and g are enough to determine such a canonical isomorphism, but the homotopy h is necessary to give stability to the concept and to construct reductions from other reductions (see the key instrument called *Basic Perturbation Lemma* in [2]).

To prove in ACL2 the Normalization Theorem, a data structure called *simplicial polynomial* was introduced in [7]. Before defining simplicial polynomials, let us start with an example. Consider $\partial_2^6\eta_3^5\partial_2^6\eta_5^5\eta_2^4\partial_1^5$, a composition of simplicial operators, defined on simplices of dimension 5. This defines a map from K_5 to K_5. First note that once we know the dimension on which it is applied, the superindexes are completely determined, so we can omit them. Second, note that if we apply the simplicial identities as rewriting rules, applied from left to right, we obtain an unique *canonical form* [1]: $\eta_4\eta_2\partial_1\partial_3$. In general, every composition of simplicial operators can be written as an equivalent expression consisting of a strictly decreasing sequence (w.r.t. its subindexes) of degeneracies and a strictly increasing sequence of faces. This canonical form is what we call a *simplicial term* and we represent it in ACL2 as a pair of lists of natural numbers, the first strictly decreasing and the second strictly increasing (in our example, `((4 2) (1 3))`).

[2]In our general definition of chain complex, the subindex ranges over $\mathbb{Z}$, so it is necessary to complete this definition with null groups and differentials in negative degrees.

In general, simplicial terms represent maps from K_n to K_m, where $m - n$ is called the *degree* of the term. The set of simplicial terms is endowed with a binary operation (composition): first, concatenate the terms and then reduce to canonical form. Thus, simplicial terms form a monoid.

Next, we can consider the *monoid ring* of the simplicial terms monoid over the integers $\mathbb{Z}$. This is the ring of *simplicial polynomials*. In ACL2, polynomials are stored in a *canonical form*: monomials have non-zero coefficients and they are sorted with respect to a total order on terms.[3] If all the terms in a simplicial polynomial represent maps from K_n to K_{n+r}, then the polynomial defines a linear map from $C_n(K)$ to $C_{n+r}(K)$, for any simplicial set K. This is the tool we used to prove in ACL2 the Normalization Theorem [7], by using polynomial expressions to describe the morphisms f, g and h in a reduction.

Our next observation was that the same tool could be useful in proving the correctness of other simplicial theorems, such as the Eilenberg-Zilber Theorem. We need the definitions of *Cartesian product* (of two simplicial sets) and of *tensor product* (of two chain complexes), in order to state that theorem.

Definition 4. *Given two simplicial sets K^1 and K^2, their* Cartesian product *is a new simplicial set, denoted by $K^1 \times K^2$, such that $(K^1 \times K^2)_n = K_n^1 \times K_n^2$ and faces and degeneracies, denoted as $\partial^\times$ and $\eta^\times$, are defined in a natural way: $\partial_i^\times(a, b) = (\partial_i a, \partial_i b)$ and $\eta_i^\times(a, b) = (\eta_i a, \eta_i b)$, respectively.*

The tensor product of two chain complexes can be defined in a general way, but in the following definition we focus on the tensor product of two *freely generated* chain complexes, the only case of application in our problem (because chain complexes associated to simplicial sets are freely generated).

Definition 5. *Given two freely generated chain complexes $C^1 := \{(C_n^1, d_n^1)\}_{n\in\mathbb{Z}}$ and $C^2 := \{(C_n^2, d_n^2)\}_{n\in\mathbb{Z}}$ (in other words, C_n^1 and C_n^2 are freely generated Abelian groups for all $n \in \mathbb{Z}$), the* tensor product *of C^1 and C^2, denoted by $C^1 \otimes C^2$, is the chain complex defined as follows. The groups $(C^1 \otimes C^2)_n$ are defined by the formula $(C^1 \otimes C^2)_n = \bigoplus_{p+q=n} C_p^1 \otimes C_q^2$, with $C_p^1 \otimes C_q^2$ the free abelian group generated by the pairs (x_p, y_q) (denoted $x_p \otimes y_q$), where x_p (y_q) ranges over the generators of C_p^1 (of C_q^2, respectively). Differentials are defined by $d_n^\otimes(x_p \otimes y_q) = d_p^1(x_p) \otimes y_q + (-1)^p x_p \otimes d_q^2(y_q)$ over generators[4], and then extended linearly over elements of $(C^1 \otimes C^2)_n$.*

And, now, the statement:

Theorem 1 (Eilenberg-Zilber reduction). *Given two simplicial sets K^1 and K^2, there exists a reduction $C(K^1 \times K^2) \Longrightarrow C(K^1) \otimes C(K^2)$.*

Since we want to formalize, and execute, a proof of this theorem in ACL2, it will be necessarily a *constructive*

[3]Kenzo also stores combinations in a canonical form (ordered with respect to a total order over the set of generators in each dimension), in order to improve the efficiency of the operations among them.

[4]The operator $\otimes$ has been overloaded to denote its linear extension for combinations.

proof, providing explicitly the triple of the reduction[5]. Then, we need to consider mappings with the shape $t : \mathbb{Z}[K_p^1 \times K_q^2] \to \mathbb{Z}[K_{p'}^1 \times K_{q'}^2]$, where K^1 and K^2 are simplicial sets. Following the same ideas as above, we can represent such a transformation t as a polynomial over *pairs* of simplicial terms. We called it a *bivariate simplicial polynomial*, and we explain the formalization of that notion in ACL2 in the next section.

3. BIVARIATE SIMPLICIAL POLYNOMIALS

As we have said, the basic components of the reduction homomorphisms in the Eilenberg-Zilber theorem are mappings from $\mathbb{Z}[K_p^1 \times K_q^2]$ to $\mathbb{Z}[K_{p'}^1 \times K_{q'}^2]$. More concretely, these morphisms can be expressed as linear combinations of pairs of compositions of faces and degeneracies. To have a faithful formalization of the standard presentation of the theorem, these morphisms will have to be defined as functions in the ACL2 logic (and in fact that will be our approach in Subsection 5.2). Nevertheless, it turns out that most of the reasoning applied to prove the theorem is carried out viewing those pairs of compositions of simplicial operators (and their linear combinations) as symbolic expressions, and operating on them following certain rules derived from the simplicial identities. This is the point of view we adopt in this section, where we present what we call *bivariate simplicial polynomials*[6], linear combinations of pairs of compositions of simplicial operators; they are a representation of morphisms as symbolic expressions built using lists and natural numbers. In ACL2, the bivariate simplicial polynomials are stored in a canonical form: every component has to be a non-zero coefficient and they are sorted with respect to a total order on pairs of simplicial terms. We have defined the function `psp-p` that checks if an expression is a bivariate simplicial polynomial in this canonical form. For example, as the simplicial term $\eta_3\partial_1\partial_2$ is represented by the two element list `((3) (1 2))` and the simplicial term $\eta_1\partial_0$ by the list `((1) (0))`, then the pair of simplicial terms $(\eta_3\partial_1\partial_2, \eta_1\partial_0)$ is represented by the two element list `(((3) (1 2)) ((1) (0)))`, and the linear combination of pairs of simplicial terms $\boldsymbol{q_1} = 3\cdot(\eta_3\partial_1\partial_2, \eta_1\partial_0) - 2\cdot(\eta_4\eta_2\partial_3, \partial_0\partial_1)$ by the list `((3 (((3) (1 2)) ((1) (0))))) (-2 (((4 2) (3)) (() (0 1)))))`.

We define componentwise the composition of pairs of simplicial terms. Then, we also define on polynomials the operations of addition, composition and scalar (integer) product, formalizing the corresponding operations on the functions they represent. For example, the composition of $\boldsymbol{q_1}$ above and $2\cdot(\eta_2\partial_1, \eta_0\partial_1) - (\eta_4\eta_2, \partial_0\partial_1)$ is the polynomial $6\cdot(\eta_3\partial_1\partial_2, \eta_1\partial_1) - 3\cdot(\eta_3\eta_2\partial_1, \eta_1\partial_0\partial_1\partial_2) - 4\cdot(\eta_4\eta_2\partial_1, \partial_0\partial_1) + 2\cdot(\eta_5\eta_4\eta_2, \partial_0\partial_1\partial_2\partial_3)$, a result we obtain applying composition of pair of simplicial terms, distributing with respect to the sums and obtaining again a linear combination in canonical form. We defined in ACL2 three functions `add-psp-psp`, `cmp-psp-psp` and `scl-prd-psp`, respectively implementing addition, composition and scalar product on simplicial polynomials. We will denote these operations as $\boldsymbol{p_1} + \boldsymbol{p_2}$, $\boldsymbol{p_1} \cdot \boldsymbol{p_2}$ and $k \cdot \boldsymbol{p}$, respectively. We also denote $\mathbf{0}$ the zero polynomial (represented by the empty list in ACL2) and $\boldsymbol{id}$ the identity polynomial (that is a polynomial with only one pair of terms; each of these terms has empty lists of faces and degeneracies).

If we denote by $\mathcal{P}^\times$ the set of bivariate simplicial polynomials (that is, the set characterized by the function `psp-p`), we proved in ACL2 that $(\mathcal{P}^\times, +, \cdot)$ is a ring, with $\mathbf{0}$ being its identity with respect to addition and $\boldsymbol{id}$ the identity with respect to composition. For example, this is one of the properties proved, establishing right distributivity[7]:

THEOREM: `cmp-psp-psp-add-psp-psp-distributive-r`
$(\boldsymbol{p}_1 \in \mathcal{P}^\times \wedge \boldsymbol{p}_2 \in \mathcal{P}^\times \wedge \boldsymbol{p}_3 \in \mathcal{P}^\times)$
$\to \boldsymbol{p}_1 \cdot (\boldsymbol{p}_2 + \boldsymbol{p}_3) = (\boldsymbol{p}_1 \cdot \boldsymbol{p}_2) + (\boldsymbol{p}_1 \cdot \boldsymbol{p}_3)$

Some of these ring properties are not trivial to prove due to the fact that all these operations return its result in canonical form (see details in [8]). Nevertheless, note that the main advantage of requiring canonical forms is that we easily can check if two given polynomials represent the same function: just check if they are syntactically equal.

4. DERIVATIVES

We can use the bivariate simplicial polynomials to formalize the maps in the proof of the Eilenberg-Zilber Theorem. For instance, we can define the differential in the Cartesian product. First, let $\boldsymbol{\partial}_i^\times$ denote the pair of simplicial terms (∂_i, ∂_i) considered as a particular case of polynomial. Then, we can introduce the following function `cartesian-diff` that recursively defines $\boldsymbol{d}_n^\times$, the polynomial representing the differential in the Cartesian product[8]:

DEFINITION: $[\boldsymbol{d}_n^\times]$
`cartesian-diff`$(n) :=$
if $n \notin \mathbb{N}^+$ **then** $\boldsymbol{\partial}_0^\times$
else $(-1)^n \cdot \boldsymbol{\partial}_n^\times$ + `cartesian-diff`$(n-1)$

For example, $\boldsymbol{d}_3^\times$ is the polinomial $(\partial_0, \partial_0) - (\partial_1, \partial_1) + (\partial_2, \partial_2) - (\partial_3, \partial_3)$.

The most difficult morphism in the Eilenberg-Zilber Theorem is the one corresponding to h, the homotopy. Due to historical reasons this arrow is called *Shih* morphism [14]. To define polynomials representing *Shih*, we are going to define an operation on polynomials called *derivative* [12]. Given a simplicial term $\eta_{i_1} \ldots \eta_{i_k} \partial_{j_1} \ldots \partial_{j_l}$, its derivative is the simplicial term obtained incesing by one the indexes of its operators, that is: $\eta_{i_1+1} \ldots \eta_{i_k+1} \partial_{j_1+1} \ldots \partial_{j_l+1}$. This operation is extended componentwise to pairs of simplicial terms, and by linearity, to polynomials. In our formalization, `derivative-psp`$(\boldsymbol{p})$ implements the derivative of a psp $\boldsymbol{p}$ (we will denote it as $\boldsymbol{p}'$). We can prove that the derivative is coherent regarding the operations of the polynomial ring, as stated by the following properties:

[5]There are more general versions of the Eilenberg-Zilber theorem (see [10]); but the formalization of those general versions seems harder than the task we undertake in our work

[6]Multivariate polynomials would appear if we extend the Eilenberg-Zilber reduction to any number of factors: $C(K^1 \times \ldots \times K^m) \Longrightarrow C(K^1) \otimes \ldots \otimes C(K^m)$. The generalization is straightforward, even if the formalization could become a bit cumbersome.

[7]In this and other statements, we adapt notations for the sake of readability, but they are the exact translation of ACL2 expressions; see [8].

[8]Note the expression between square brackets in the first line of the definition; in general, this will be the way we will show how a function will be denoted subsequently.

THEOREM: `derivative-psp-add-psp-psp`
$\boldsymbol{p}_1 \in \mathcal{P}^\times \wedge\ \boldsymbol{p}_2 \in \mathcal{P}^\times \to (\boldsymbol{p}_1 + \boldsymbol{p}_2)' = \boldsymbol{p}_1' + \boldsymbol{p}_2'$

THEOREM: `derivative-psp-cmp-psp-psp`
$\boldsymbol{p}_1 \in \mathcal{P}^\times \wedge\ \boldsymbol{p}_2 \in \mathcal{P}^\times \to (\boldsymbol{p}_1 \cdot \boldsymbol{p}_2)' = \boldsymbol{p}_1' \cdot \boldsymbol{p}_2'$

THEOREM: `derivative-psp-scl-prd-psp`
$\boldsymbol{p} \in \mathcal{P}^\times \wedge\ k \in \mathbb{Z}\ \to (k \cdot \boldsymbol{p})' = k \cdot \boldsymbol{p}'$

To prove these properties, we have to deal with how adding one to the subindexes of the simplicial operators affects the ring operations, taking into account that the polynomials are in canonical form. In particular, to prove the property `derivative-psp-cmp-psp-psp`, we needed two main lemmas.

First, we showed that the derivative distributes over composition of simplicial terms; recall that when we compose simplicial terms, they are returned in canonical form, and that canonical form is obtained exhaustively rewriting with the simplicial identities. Thus, the property on terms is a consequence of the fact that the simplicial identities are preserved if we add one to the subindexes. The second lemma deals with the ordering on terms needed for a polynomial to be considered in canonical form. We proved that this ordering is preserved by the derivative operation.

Note that these two lemmas were "suggested" by the ACL2 theorem prover, in a way we will explain now. Although the prover is automatic in the sense that there is no user interaction once a proof attempt starts, we can say that ACL2 is interactive in a wider sense. If a proof attempt fails, one can inspect the output, trying to guess in which point the prover digresses from the intended proof one has in mind. Usually this happens because it needs previously proved results that, used as simplification rewrite rules, would lead the prover to a successful proof.

In the case of the derivative of a composition, the two main lemmas mentioned above were suggested by inspecting failed proof attempts. In turn, to prove these two lemmas, several sublemmas were suggested about how derivatives affect to the canonical form of a simplicial term. Carrying out proofs in this way, the user provides the prover a collection of lemmas that finally let it to prove the intended lemma, only using induction and simplification. This is a standard way to interact with ACL2, called *The Method* [6] by the authors of the system. In the development of the formal proof of the Eilenberg-Zilber theorem, we followed *The Method*.

Finally, having defined derivatives, the function `SH-pol`(n) obtains the corresponding polynomial representing the *Shih* homomorphism in dimension n:

DEFINITION: $[\boldsymbol{SH}_n]$
`SH-pol`$(n) :=$
if $n \notin \mathbb{N}^+$ **then 0**
else $-1 \cdot ((\texttt{SH-pol}(n-1))' + (\sum_{i=0}^{n} \boldsymbol{EML}_{n-i,i} \cdot \boldsymbol{AW}_{n,i})' \cdot \boldsymbol{\eta}_0^\times)$

where $\boldsymbol{AW}$ and $\boldsymbol{EML}$ are the polynomials associated with the morphisms f and g, respectively, in the Eilenberg-Zilber reduction (see [14] for details), and $\boldsymbol{\eta}_0^\times$ is the polynomial (η_0, η_0).

5. EXECUTING AND EVALUATING POLYNOMIALS

Simplicial polynomials can be dealt with in two ways: as symbolic expressions, which can be handled by means of algorithms; and as codes for morphisms, which can be evaluated over chains of simplices. We will see now that both kinds of manipulations are useful for certified computing.

5.1 Simplifying and executing

The first point of view is very useful when proving properties, as we showed in the previous section. Identities between simplicial polynomials can be used as simplification rules for ACL2, in such a way that automation can be increased in proofs.

But we can use simplicial polynomials to help proofs in another way. Because ACL2 is also a programming environment, we can *execute* the recursive definitions, as that of $\boldsymbol{SH}$ in the previous section. Then we can also execute the expressions involved in statements. This could give us experimental insights about how proving the theorem, as well as guide us to the statement of new lemmas needed in the development. In fact, it has been the case in several of the most complicated parts of the mechanized proof of the Eilenberg-Zilber theorem. Let us illustrate these ideas with an example, only involving the *Shih* operator.

First, we can compute the explicit expression of $\boldsymbol{SH}$ for some (small) values of the dimension n.

```
ACL2 !>(sh-pol 4)
((-1 (((0) ()) ((4 3 2 1) (1 2 3))))
 (1 (((1) ()) ((4 3 2) (2 3))))
 (1 (((1 0) (4)) ((4 3 2) (1 2))))
 (-1 (((2) ()) ((4 3) (3))))
 (-1 (((2 0) (4)) ((4 3 1) (1 2))))
 (1 (((2 1) (4)) ((4 3) (2))))
 (-1 (((2 1 0) (3 4)) ((4 3) (1))))
 (1 (((3) ()) ((4) ())))
 (1 (((3 0) (4)) ((4 2 1) (1 2))))
 (-1 (((3 1) (4)) ((4 2) (2))))
 (1 (((3 1 0) (3 4)) ((4 2) (1))))
 (1 (((3 2) (4)) ((4) ())))
 (-1 (((3 2 0) (3 4)) ((4 1) (1))))
 (1 (((3 2 1) (3 4)) ((4) ())))
 (1 (((3 2 1 0) (2 3 4)) ((4) ())))
 (-1 (((4 0) (4)) ((3 2 1) (1 2))))
 (1 (((4 1) (4)) ((3 2) (2))))
 (-1 (((4 1 0) (3 4)) ((3 2) (1))))
 (-1 (((4 2) (4)) ((3) ())))
 (1 (((4 2 0) (3 4)) ((3 1) (1))))
 (-1 (((4 2 1) (3 4)) ((3) ())))
 (-1 (((4 2 1 0) (2 3 4)) ((3) ())))
 (1 (((4 3) (4)) ((3) ())))
 (-1 (((4 3 0) (3 4)) ((2 1) (1))))
 (1 (((4 3 1) (3 4)) ((2) ())))
 (1 (((4 3 1 0) (2 3 4)) ((2) ())))
 (-1 (((4 3 2) (3 4)) ((2) ())))
 (-1 (((4 3 2 0) (2 3 4)) ((1) ())))
 (1 (((4 3 2 1) (2 3 4)) ((1) ())))
 (-1 (((4 3 2 1 0) (1 2 3 4)) ((0) ()))))
```

Note that some of the pairs in the result are representing *degenerate* operators, that is, they produce degenerate simplices when they are evaluated over chains of simplices. In the Cartesian product, this means that they have

a common index in the degeneracy list of each simplicial term in the pair. With our syntax, they are detected because there is a common integer in the first list of each simplicial term. For instance, in the last simplicial term `(((4 3 2 1 0) (1 2 3 4)) ((0) ()))` the index 0 is present in `(4 3 2 1 0)` and `(0)`. As bivariate simplicial polynomials, the degenerate pairs are kept, but they are eliminated later in a *normalization* process because the morphisms they represent always produce degenerate simplices.

By examining the previous expression, the ocurrence of *shuffles* can be expected, as foreseen in the formula presented in [14]. A (p,q)-shuffle $(\alpha_1,\ldots,\alpha_p,\beta_1,\ldots,\beta_q)$ is a permutation of the set $\{0,1,\ldots,p+q-1\}$ such that $\alpha_i<\alpha_{i+1}$ and $\beta_j<\beta_{j+1}$, when $i=1,\ldots,p-1$ and $j=1,\ldots,q-1$. Let us look for such shuffles in the two first monomials of the previous expression. The first monomial is `(-1 (((0) ()) ((4 3 2 1) (1 2 3))))`. After the coefficient -1, we have a pair of terms: the first one is `((0) ())`, denoting a term with only a degeneracy η_0 and no faces; and the second one is `((4 3 2 1) (1 2 3))` with $\eta_4\eta_3\eta_2\eta_1$ in the degenerate part. Here we find the $(1,4)$-shuffle $((0),(1,2,3,4))$. In the second monomial, `(1 (((1) ()) ((4 3 2) (2 3))))`, we find the *first derivative* (the level of derivation is determined by the smallest degenerate index in the first term) of the $(1,3)$-shuffle $((0),(1,2,3))$. By carefully examining the results of execution, new organizations of the formula presented in [14] can be found, that can be helpful when proving some properties of morphisms.

For instance, the morphism *Shih* must satisfy condition (5) in the definition of a reduction (that is to say, $h\circ h=0$). Since the Eilenberg-Zilber theorem is stated in terms of the *normalized* chain complex, this means that the composition of $\boldsymbol{SH}$ with itself must give always degenerate terms. We can compute that composite for particular cases:

```
ACL2 !>(cmp-psp-psp (sh-pol 3) (sh-pol 2))
((-1 (((2 1 0) (2)) ((3 1) ())))
 (1 (((3 1 0) (2)) ((2 1) ())))
 (-1 (((3 1 0) (2)) ((3 2) ())))
 (-1 (((3 2 0) (2)) ((2 1) ())))
 (1 (((3 2 0) (2)) ((3 1) ())))
 (1 (((3 2 1) (2)) ((2 1) ())))
 (-1 (((3 2 1) (2)) ((3 1) ())))
 (1 (((3 2 1 0) (1 2)) ((1 0) ())))
 (-1 (((3 2 1 0) (1 2)) ((2 0) ())))
 (1 (((3 2 1 0) (1 2)) ((3 0) ()))))
```

It can be checked that, as foreseen, all the terms are denoting degenerate simplices in the Cartesian product, because there is always a common integer in the first list of each simplicial term.

Thus, by inspecting those expressions, we can *test* that the property holds, and even better, we can establish conjectures to organize the general proof of the property.

5.2 Evaluating

In order to interpret a simplicial polynomial as a morphism between chain complexes, we need to represent in ACL2 chain complexes (associated with simplicial sets) and then define how symplicial polynomials can be evaluated on chains of simplices. Finally, in order to get an executable function (corresponding to a simplicial polynomial) we need to instantiate generic simplicial sets (appearing in the proofs) to concrete ones.

Let us first deal with how we represent simplicial sets. Note that a simplicial set is characterized by a set K and families of functions (faces and degeneracies) satisfying certain properties. Since the Eilenberg-Zilber theorem is about any two simplicial sets, we have to introduce them in a completely generic way. Although in ACL2 the usual way to introduce functions in the logic is by the definition principle (using `defun`), it also provides the *encapsulation* principle (using `encapsulate`), which allows to introduce functions in the logic without defining them completely, only stating about them some assumed properties [6].

In our formalization, a generic simplicial set is defined by means of three functions `K`, `d` and `n`. The function `K` is a predicate of two arguments, with the intended meaning that `K`(n,x) holds when $x\in K_n$. Faces and degeneracies are represented, respectively, by the functions `d` and `n`, both with three arguments. The idea is that `d`(m,i,x) and `n`(m,i,x) respectively represent $\partial_i^m(x)$ and $\eta_i^m(x)$. These three functions are introduced using `encapsulate`, only assuming about them well-definedness and the simplicial identities. For example, the following are the assumptions corresponding respectively to the well-definedness of `d` and the first simplicial identity:

ASSUMPTION: `d-well-defined`
$(x\in K_m \wedge m\in\mathbb{N}^+ \wedge i\in\mathbb{N}\wedge i\le m)\rightarrow \partial_i^m(x)\in K_{m-1}$

ASSUMPTION: `simplicial-id1`
$(x\in K_m \wedge m,i,j\in\mathbb{N}\wedge j\le i\wedge i<m\wedge 1<m)$
$\rightarrow \partial_i^{m-1}(\partial_j^m(x))=\partial_j^{m-1}(\partial_{i+1}^m(x))$

We omit here the rest of the assumptions (i.e., well-definedness of `n` and the rest of the simplicial identities), since they are stated in an analogous way.

As for the formalization of chains of simplices, since they are formal linear combinations of non-degenerate simplices, it is quite natural to represent them as lists of pairs of an integer coefficient and a non-degenerate simplex. As with polynomials, we consider chains in canonical form (as *Kenzo* does; see footnote 3): we do not allow zero coefficients and we require the pairs to be increasingly ordered with respect to a strict ordering on simplices. The following function `scn-p` defines chains in a given dimension n. It uses the auxiliary functions `ssn-p`, which recognizes pairs of a non-null integer and a non-degenerate simplex, and `ssn-<` which defines the lexicographic strict order between such pairs:

DEFINITION: $[c\in C_n(K)]$
`scn-p`(n,c) :=
  if `endp`(c) **then** $c=$ **nil**
  elseif `endp`(`rest`(c))
    then `ssn-p`$(n$,`first`$(c))\wedge$ `rest`$(c)=$ **nil**
  else `ssn-p`$(n$,`first`$(c))\wedge$
    `ssn-<`$(n$,`first`(c),`second`$(c))\wedge$
    `scn-p`$(n$,`rest`$(c))$

We also define addition of chains, and the scalar product of an integer and a chain. These operations act on chains in the canonical form described above, and return chains also in canonical form. We proved that the set of chains of a given dimension is an Abelian group with respect to addition, where the identity is represented by the empty list.

Now, we have to formally specify the functional interpretation of a polynomial. That is, we define an ACL2 function such that given a polynomial and a chain of pairs of simplices of a given dimension, it computes the result of evaluating the function that the polynomial is supposed to represent, on the given chain.

First, we have to define some well-formedness conditions on polynomials. Think for example in the following simplicial term: $\eta_5\eta_1\partial_3$. This term cannot be interpreted as a function on $C_4(K)$, regardless of the simplicial set K, because in such case, η_5 would have to be applied to a simplex in $C_4(K)$, which is not possible. Nevertheless, it makes sense to apply it to any chain of dimension $n \geq 5$. We will say that a simplicial term is *valid for dimension* m, when interpreted as composition of simplicial operators, can be applied to any simplex of dimension m. Another notion to take into account is what we called previously the *degree* of a term: if a term is valid for n and it represents a function from K_n to K_m, its degree is $m - n$ (for example, the degree of the previous term is 1). Extending these concepts to pairs, we will say that a pair of simplicial terms (t_1, t_2) is valid for dimension (m_1, m_2) with degree (j_1, j_2) if t_i is valid for m_i and with degree j_i $(i = 1, 2)$. We say that a polynomial is *well-formed for dimension* (m_1, m_2) if all its terms are valid for that dimension and with the same degree. The *degree* of a polynomial is the common degree of its terms.

Well-formed polynomials for dimension (m_1, m_2) represent valid morphisms whose evaluation can be defined on $\mathbb{Z}[K^1_{m_1} \times K^2_{m_2}]$, where K^1 and K^2 are simplicial sets. We have defined in ACL2 a function `eval-psp`($\boldsymbol{p}$,m_1,m_2,c) that computes the result of evaluating a polynomial $\boldsymbol{p}$ on a linear combination c of pairs of simplices of dimension (m_1, m_2). This function does not remove degenerate simplices because this property is different if the result is in the Cartesian product or in the tensor product, but the evaluation is the same in both. We also proved that `eval-psp` is a homomorphism on the ring of polynomials. For example, under the corresponding well-formedness conditions, the evaluation of the composition of two polynomials is equal to the composition of the evaluations of the polynomials, and analogously for addition and scalar product. This is proved in a similar way as it is described in [7] for "univariate" simplicial polynomials.

Finally the definition of the SH function from the corresponding polynomial is as follows (SH_n is a function from $C_n(K^1 \times K^2)$ to $C_{n+1}(K^1 \times K^2)$). We can prove that the polynomial $\boldsymbol{SH}_n$ defined at the end of Section 4, is well-formed for dimension (n, n), with degree $(1, 1)$. So it is valid to define SH_n on a given chain, as first evaluating $\boldsymbol{SH}_n$ on the chain and then eliminate degenerate addends with respect to the Cartesian product (the function `Fx-norm` is in charge of eliminating these degenerate elements)

DEFINITION: $[SH_n(c)]$
`SH`(n,c) := `Fx-norm`($n+1$,`eval-psp`($\boldsymbol{SH}_n$,n,n,c))

Therefore, we have obtained a function SH from the simplicial polynomial associated with the morphism *Shih*. Nevertheless, it is not still executable, because proofs are carried out on *generic* simplicial sets, ensuring the properties are true for any two simplicial sets. In order to get running examples, we instantiate the previous generic theory over two concrete simplicial sets, proving all the assumptions that have been set on the generic simplical sets. As an example, we use the *standard simplex* Δ [10]. This simplicial set has some universal properties, since the simplicial identities are the unique constraints in it. In particular, any generic formula relating simplicial equalities will be faithfully drawn on Δ (see [10]). The definition of this simplicial set is the following: simplices in Δ are non-decreasing lists of natural numbers (lists of length $n+1$ if we are in dimension n); a face of index i consists in erasing the element at position i; and a degeneracy of index i consists in repeating the element at position i in the list.

5.3 An example

Once the proof of the Eilenberg-Zilber theorem has been instantiated on the standard simplex Δ, we can *run* the different morphisms. We concentrate on the *Shih* morphism, being the more complex one. Furthermore, we can make computing *Kenzo* in the same examples, and then compare both results. Even if the data structures used in our implementation are different from that of *Kenzo*, and consequently the algorithms are also different, in essence both implementations are based on the same definition of the triple of the reduction.

The test is running over the Cartesian product $\Delta \times \Delta$, and then applied over the chain with only one component, with coefficient 1 and generator $((0, 1, \ldots, n), (0, 1, \ldots, n))$, belonging to $C_n(\Delta \times \Delta)$. Next we include the chain obtained by ACL2, in the case $n = 4$.

```
ACL2 !>(SH 4 (Delta1 4))
((1 ((0 0 0 0 0 1) (0 1 2 3 4 4)))
 (-1 ((0 0 0 0 1 1) (0 1 2 3 3 4)))
 (-1 ((0 0 0 0 1 2) (0 2 3 4 4 4)))
 (1 ((0 0 0 1 1 1) (0 1 2 2 3 4)))
 (1 ((0 0 0 1 1 2) (0 2 3 3 4 4)))
 (-1 ((0 0 0 1 2 2) (0 2 3 3 3 4)))
 (1 ((0 0 0 1 2 3) (0 3 4 4 4 4)))
 (-1 ((0 0 1 1 1 1) (0 1 1 2 3 4)))
 (-1 ((0 0 1 1 1 2) (0 2 2 3 4 4)))
 (1 ((0 0 1 1 2 2) (0 2 2 3 3 4)))
 (-1 ((0 0 1 1 2 3) (0 3 3 4 4 4)))
 (-1 ((0 0 1 2 2 2) (0 2 2 2 3 4)))
 (1 ((0 0 1 2 2 3) (0 3 3 3 4 4)))
 (-1 ((0 0 1 2 3 3) (0 3 3 3 3 4)))
 (-1 ((0 0 1 2 3 4) (0 4 4 4 4 4)))
 (1 ((0 1 1 1 1 2) (0 1 2 3 4 4)))
 (-1 ((0 1 1 1 2 2) (0 1 2 3 3 4)))
 (1 ((0 1 1 1 2 3) (0 1 3 4 4 4)))
 (1 ((0 1 1 2 2 2) (0 1 2 2 3 4)))
 (-1 ((0 1 1 2 2 3) (0 1 3 3 4 4)))
 (1 ((0 1 1 2 3 3) (0 1 3 3 3 4)))
 (1 ((0 1 1 2 3 4) (0 1 4 4 4 4)))
 (1 ((0 1 2 2 2 3) (0 1 2 3 4 4)))
 (-1 ((0 1 2 2 3 3) (0 1 2 3 3 4)))
 (-1 ((0 1 2 2 3 4) (0 1 2 4 4 4)))
 (1 ((0 1 2 3 3 4) (0 1 2 3 4 4))))
```

Let us interpret some of the components of this expression. The first one is `(1 ((0 0 0 0 0 1) (0 1 2 3 4 4)))`. That means that the operator $(\eta_3\eta_2\eta_1\eta_0\partial_2\partial_3\partial_4, \eta_4)$ has been applied to `((0 1 2 3 4) (0 1 2 3 4))`. And this corresponds to the bivariate simplicial monomial `(1 (((3 2 1 0) (2 3 4)) ((4) ())))` which appears in the expression obtained by executing `(sh-pol 4)` in Subsection 5.1. It is the same for the rest of monomials, except

for the degenerate ones, that, as announced in 5.1, have been cancelled during the normalization process. Of course, the results are also coherent in all dimensions in which the testing has been carried out.

The result of executing `(SH-n 4 (Delta1 4))` in ACL2 is also equal (up to combinations representation) to the one obtained with *Kenzo* (and it has been always the case in all the *automated* test cases we have checked). In addition, the implementation in *Kenzo* was based in the explicit formulas from [14]. As explained in Subsection 5.1, these formulas nicely correspond with the bivariate simplicial polynomials defined in Section 4. And it is from evaluating this polynomials how we have obtained the ACL2 executable morphisms. The coherence of these four components in this architecture gives a solid evidence that the formulas appearing in our constructive proof of the Eilenberg-Zilber theorem are the same implemented in *Kenzo*. As a consequence, the confidence in the correctness of *Kenzo* is reinforced.

6. CONCLUSIONS

In this paper, we have illustrated the role of symbolic manipulation in a context of certified computing. Even if the case study is taken from a rather specialized area (namely, Computational Simplicial Topology), we hope that the consequences of our methods can be useful for a range of researchers, working in automated reasoning, in algebraic computing or, at it is our case, in the frontier between both disciplines.

The main technical contribution of the paper is the introduction of *bivariate simplicial polynomials*, a data structure which allows us to enhance the ACL2 theorem prover with a kind of *algebraic* rewriting, greatly increasing the automation of the proofs.

Simplicial polynomials can be viewed from two different perspectives. In the first one, a simplicial polynomial is representing symbolically a family of natural transformations; interestingly enough, this allows us a kind of *logical* reduction: representing some higher-order constructs (i.e. natural transformations between functors) as first-order elements (lists of integers); recall that ACL2 is a first-order tool.

From the second point of view, a simplicial polynomial can be used to define morphisms between chain complexes, by means of a evaluation of the polynomial over chains of simplices.

In both perspectives executability is important (ACL2, as some other proof assistants, allows running definitions and expressions). In the first one, unfolding recursive definitions of polynomials was useful for conjecturing some lemmas which guided the proof of the main theorems. In the second perspective, executing morphisms (on concrete simplicial sets) permits us an *automated* testing of the *Kenzo* program: *Kenzo* results were confronted to their ACL2 verified counterpart.

Verifying technology is still too poor to undertake the complete correctness analysis of a software system as complex as *Kenzo*. So, we approach the problem with several tools and apply a separation of concerns strategy: sometimes we use proof assistants to verify the correctness of *algorithms* and in other occasions we concentrate on verifying actual running code. In this paper, we focused on a concrete result in Simplicial Topology: the Eilenberg-Zilber theorem, showing the feasibility and usefulness of our proposals.

As for further work, there is many room for extensions and improvements. Let us mention simply the possibility of using ACL2 tools (compilation, guards, single-threaded objects, and so on; see [6]) to speed up computation in the certified side; then the testing of *Kenzo* would be more complete, and so the confidence in its correctness would be still more reinforced.

7. REFERENCES

[1] M. Andrés, L. Lambán, J. Rubio, and J.-L. Ruiz-Reina. Formalizing Simplicial Topology in ACL2. *Proceedings ACL2 Workshop 2007*, 34–39, 2007.

[2] J. Aransay, C. Ballarin, and J. Rubio. A Mechanized Proof of the Basic Perturbation Lemma. *Journal of Automated Reasoning*, 40(4):271–292, 2008.

[3] F. Boudaoud, F. Caruso, and M.F. Roy. Certificates of Positivity in the Bernstein Basis. *Discrete and Computational Geometry*, 39(4):639–655, 2008.

[4] C. Domínguez, and J. Rubio. Effective homology of bicomplexes, formalized in Coq. *Theoretical Computer Science*, 412(11):962–970, 2011.

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

[6] M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer, 2010.

[7] L. Lambán, F.J. Martín-Mateos, J. Rubio, and J.-L. Ruiz-Reina. Formalization of a normalization theorem in simplicial topology. *Annals of Mathematics and Artificial Intelligence* 64(1):1–37, 2012.

[8] L. Lambán, F.J. Martín-Mateos, J. Rubio, and J.-L. Ruiz-Reina. Formalization of the Eilenberg-Zilber Theorem, 2012. `http://www.glc.us.es/fmartin/acl2/eztheorem`

[9] F.J. Martín-Mateos, J. Rubio, and J.-L. Ruiz–Reina. ACL2 verification of simplicial degeneracy programs in the Kenzo system, *Proceedings Calculemus 2009, Lecture Notes in Artificial Intelligence*, 5625:106–121, 2009.

[10] J. P. May. *Simplicial objects in Algebraic Topology*. Van Nostrand, 1967.

[11] G. C. Necula, and P. Lee. Proof-Carrying Code. Technical Report CMU-CS-96-165, 1996.

[12] P. Real. Homological Perturbation Theory and Associativity. *Homology, Homotopy and Applications* 2(5):51–88, 2000.

[13] A. Romero, and J. Rubio. Homotopy groups of suspended classifying spaces: an experimental approach. To appear in Mathematics of Computation, 2013.

[14] J. Rubio. Homologie effective des espaces de lacets itérés : un logiciel. Thèse, Institut Fourier, 1991. `http://dialnet.unirioja.es/servlet/tesis?codigo=1331`