Verifying the bridge between Simplicial Topology and Algebra: the Eilenberg-Zilber algorithm*

L. Lambán[1] F. J. Martín–Mateos[2] J. Rubio[1]
J. L. Ruiz–Reina[2]
[1] Dept. of Mathematics and Computation,
University of La Rioja,
Edificio Vives, Luis de Ulloa s/n.
26004 Logroño, Spain
[2] Computational Logic Group,
Dept. of Computer Science and Artificial Intelligence,
University of Seville, Avda. Reina Mercedes, s/n.
41012 Sevilla, Spain

Abstract

The Eilenberg-Zilber algorithm is one of the central components of the computer algebra system called *Kenzo*, devoted to computing in Algebraic Topology. It this paper we report on a complete formal proof of the underlying Eilenberg-Zilber *theorem*, using the ACL2 theorem prover. As our formalization is executable, we are able to compare the results of the certified programs with those of *Kenzo* on some *universal* examples. Since the results coincide, the reliability of *Kenzo* is so reinforced. This is a new step in our long term project towards certified programming for Algebraic Topology.

1 Introduction

Nowadays, computing in Algebraic Topology is having an increasing importance, both in pure and in applied mathematics [6]. In this area, the *Kenzo* system [5] has some particular features. *Kenzo* is a Common Lisp program, created by F. Sergeraert, that can deal with infinite dimensional spaces, and is able to compute results that have not been determined by any other mean (theoretical or computational). In [20] a theorem corrected thanks to *Kenzo* is presented, together with other results computed with *Kenzo* which seem out of the reach for any other method. In addition, *Kenzo* has been instrumental in some computations related to the homological processing of biomedical images, in an on-going project on drug design against Alzheimer [8].

Due to these features of *Kenzo*, 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. A fundamental algorithm implemented in *Kenzo* (known as *Basic Perturbation Lemma*) has been formalized in Isabelle/HOL [2, 3]. Also the Coq proof assistant has

*This work was partially supported by Ministerio de Ciencia e Innovación, [MTM2009-13842], and by European Union's 7th Framework Programme [243847] (ForMath).

been used, for instance, to model the homology of bicomplexes [4] and the computation of homology groups in the finite case [7].

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. It was therefore natural to use ACL2 [10], a theorem prover intimately linked to Common Lisp. Even though ACL2 is not suitable to model all *Kenzo* characteristics (as *Kenzo* uses higher-order functional programming, while ACL2 is rather a first-order tool; this explains the role of Isabelle/HOL and Coq in our global project), it is superior to any other tool when formalizing the *actual Kenzo* source code.

In this area of ACL2 application to Algebraic and Simplicial Topology, several objectives has been already achieved. In [1] simplicial sets where studied as rewriting systems, producing in a new way the canonical decomposition of each simplex. The degeneracy encoding used in *Kenzo* was formalized and proved correct in [15]. Finally, a *normalization theorem* needed as a preprocessing justifying the *Kenzo* way of working was also proved in ACL2 [12].

The formal proof of the Normalization Theorem was carried out by using a conceptual tool called *simplicial polynomial* [11], which allows us to enhance ACL2 with a kind of *algebraic rewriting* (namely, a simplification strategy for rings), providing a greater automation in the proof. This same tool is now applied to the correctness proof of the Eilenberg-Zilber algorithm. It is not only reused at the conceptual level, but it also provides a truly *proof reusing*, as illustrated in the following figures. The Eilenberg-Zilber Theorem needed around 13000 lines of ACL2 code [13], while the Normalization Theorem needed around 4500 lines [12] (so, Eilenberg-Zilber can be considered three times more difficult than normalization). These data must be however be tempered with the existence of 6000 lines of ACL2 code devoted to infrastructure (algebraic rewriting, meta-rules, macro and theory generating facilities, and so on; see [12] for details). This infrastructure was prepared for the Normalization Theorem and then it has been fully reused in the Eilenberg-Zilber formalization. Thus, the learned lesson is that paying attention to a systematic development can be rewarding in mechanized theorem proving (as it is in computer programming).

As for the conceptual importance of the Eilenberg-Zilber Theorem, let us explain roughly it establishes the bridge between geometry and algebra in Algebraic (Simplicial) Topology. More concretely, it states a homological equivalence between (the chain complex of) a Cartesian product (a geometric construction) and the tensor product of two chain complexes (a purely algebraic construction). This fundamental aspect of Eilenberg-Zilber is reflected in its computational counter-part: experimental studies of log files for *Kenzo* showed that most of the running time is devoted to Eilenberg-Zilber computations (and more concretely to the computation of the map 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 hard formal methods in computer algebra verification. In addition, we hope that the different technical tools introduced in ACL2 to deal with complex combinatorial structures can be of help in the area of automated theorem proving.

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 formal proof, introducing a generalization of simplicial polynomials, some ACL2 enhancements and a short presentation of the ACL2 code (for details, the reader is referred to [13]). Then, in Section 4 the aspects related to executability are developed; it includes an operational interpretation of the proof, its application to a *universal* example and the comparison with the real *Kenzo* results, showing the coherence and usefulness of our approach. The paper ends with conclusions, further work and the bibliography.

2 Statement of the mathematical problem

In this section we introduce briefly the notions needed to state the main theorem (for details and context about Simplicial Topology, see, for instance, [16]).

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

$$\begin{array}{lll} \partial_i^n : K_n \to K_{n-1}, & n > 0, & i = 0, \ldots, n, \\ \eta_i^n : K_n \to K_{n+1}, & n \geq 0, & i = 0, \ldots, n, \end{array}$$

subject to the following equations:

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

where id^n is the identity map on K_n.

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 Im $d_{n+1} \subseteq$ Ker d_n; then the *homology groups* of C: $H_n(C) = \text{Ker } d_n/\text{Im } d_{n+1}$ are well-defined. 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 by $C_n(K)$. That is, 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 a 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 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

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

[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.

for differentials: $d_n(x) = \sum_{i=0}^{n}(-1)^i \partial_i(x)$ (in this case, no term is erased). If we called $\widehat{C(K)}$ this (bigger) chain complex associated with a simplicial set K, it is the case that the respective homology groups corresponding to $\widehat{C(K)}$ and $C(K)$ are canonically isomorphic; this is exactly the statement of the afore-mentioned Normalization Theorem [12]. More concretely, in [12] a *reduction* $\widehat{C(K)} \Longrightarrow C(K)$ was implemented in ACL2:

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 homomorphisms such that $f \circ d^1 = d^2 \circ f$ and $g \circ d^2 = d^1 \circ g$), and h is graded morphism of degree $+1$ (called* homotopy operator*), that is to say a family of homomorphisms $h_n : C_n^1 \to C_{n+1}^1$ satisfying (1) $f \circ g = id$, (2) $d \circ h + h \circ d + g \circ f = id$, (3) $f \circ h = 0$, (4) $h \circ g = 0$, and (5) $h \circ h = 0$.*

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 respective 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 allowing one to construct reductions from other reductions (see the key instrument called *Basic Perturbation Lemma* in [2]).

We are almost ready to introduce the Eilenberg-Zilber Theorem. We still need the definitions of *Cartesian product* (of two simplicial sets) and of *tensor product* (of two chain complexes).

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, respectively denoted as $\partial^\times$ and $\eta^\times$, are defined as: $\partial_i^\times(a, b) = (\partial_i a, \partial_i b)$ and $\eta_i^\times(a, b) = (\eta_i a, \eta_i b)$.*

In the following definition of tensor product of chain complexes, we restrict ourselves to the particular case of freely generated chain complexes.

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[3], and then extended linearly over elements of $(C^1 \otimes C^2)_n$.*

And, now, the statement to be formalized in ACL2:

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

In particular, from the point of view of the explicit calculation of homology groups, the Eilenberg-Zilber theorem allows one to replace $C(K^1 \times K^2)$ by an *smaller* chain complex $C(K^1) \otimes C(K^2)$.

In fact, there is a much more explicit statement of the theorem, giving rise to an actual algorithm, because a reduction

$$(f, g, h) : C(K^1 \times K^2) \Longrightarrow C(K^1) \otimes C(K^2)$$

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

is known, where the maps f, g, and h are defined as:

$$
\begin{aligned}
f(x_n, y_n) &= \sum_{i=0}^{n} \partial_{i+1} \dots \partial_n x_n \otimes \partial_0 \dots \partial_{i-1} y_n \\
g(x_p \otimes y_q) &= \sum_{(\alpha,\beta)\in\{(p,q)\text{-shuffles}\}} (-1)^{sg(\alpha,\beta)} (\eta_{\beta_q} \dots \eta_{\beta_1} x_p, \eta_{\alpha_p} \dots \eta_{\alpha_1} y_q) \\
h(x_n, y_n) &= \sum (-1)^{n-p-q+sg(\alpha,\beta)} (\eta_{\beta_q+n-p-q} \dots \eta_{\beta_1+n-p-q} \eta_{n-p-q-1} \partial_{n-q+1} \dots \partial_n x_n, \\
&\qquad \eta_{\alpha_{p+1}+n-p-q} \dots \eta_{\alpha_1+n-p-q} \partial_{n-p-q} \dots \partial_{n-q-1} y_n)
\end{aligned}
$$

where a (p, q)-shuffle $(\alpha, \beta) = (\alpha_1, \dots, \alpha_p, \beta_1, \dots, \beta_q)$ is a permutation of the set $\{0, 1, \dots, p + q - 1\}$ such that $\alpha_i < \alpha_{i+1}$ and $\beta_j < \beta_{j+1}$, $sg(\alpha, \beta) = \sum_{i=1}^{p}(\alpha_i - i - 1)$, and the third sum (which defines the homotopy operator h) is taken over all the indices $0 \le q \le n-1, 0 \le p \le n-q-1$ and $(\alpha, \beta) \in \{(p + 1, q)\text{-shuffles}\}$.

The maps f, g, and h are known respectively as the *Alexander-Whitney* (*AW* in short), *Eilenberg-MacLane* (*EML*), and *Shih* (*SH*) operators.

It is worth stressing that the Eilenberg-Zilber Theorem, under its very concrete form of a reduction, holds with the $C(-)$ chain complex model, but it is not longer true with the bigger model $\widehat{C(-)}$. Thus, this gives a new interest to having formalized the Normalization Theorem [12] before dealing with the Eilenberg-Zilber result.

The formulas for *AW* and *EML* where classically known. The expression for *SH* was given for the first time in [21] (it was experimentally found when programming *EAT* [22], the predecessor of *Kenzo*). Then it was formally proved by F. Morace and published as an appendix for a paper by P. Real [19].

Several comments can be made about the expressions. First, they are essentially unique ([17, 18]), so in some sense they are unavoidable. Second, due to the occurrence of the *shuffles* their nature is *exponential* (at least, if dimensions are considered a size of the problem) and, in fact, this is one of the reasons why *Kenzo* performance is dramatically decreased when dimensions increase.

Although *EML* and *SH* have a quite frightening aspect, actually the expressions are very well structured and of a combinatorial nature, and these features allow us to devise a proof purely based on induction and rewriting (inspired at some points by ideas from [19]). This is the proof which has been fully formalized by using the proof assistant ACL2 [10], as described in the following sections.

3 ACL2 formalization of the EZ theorem using simplicial polynomials

In this section we describe the main aspects of the formalization of the Eilenberg-Zilber theorem in ACL2, in the framework of what we call simplicial polynomials. This is a conceptual tool already used to prove the Normalization Theorem in Simplicial Topology [12] and now extended to deal with this formalization.

ACL2 is a programming language (an extension of an applicative subset of Common Lisp), a logic to state and prove properties about the programs written in the language, and a theorem prover assisting in the task of proving the properties. The logic of ACL2 is a first-order logic, describing an extension of an applicative subset of Common Lisp. It includes logical axioms as well as axioms describing built-in functions in the language. Rules of inference include those of propositional logic, equality, instantiation and induction. The theorem prover mechanizes the logic; although every proof attempt runs automatically, the role of the user is important: a successful formalization requires the construction of a theory by means of definitions and lemmas, that once proved are used by the system in subsequent proof attempts.

We only give the main lines of our formalization and therefore many details will be omitted. Although the ACL2 syntax is the Common Lisp syntax (and in particular uses parentheses and prefix notation), in this paper, for the sake of readability, we will use a notation closer to the usual mathematical notation, and in particular some functions will be used in infix notation. The complete source files containing the ACL2 formalization and proof of the Eilenberg-Zilber theorem are available at [13]. There, we also include a complete index with the correspondence of the functions and theorems referenced in this paper with those in the formalization.

3.1 Representational issues

This section is devoted to show how the apparent complexity of some algebraic constructors (as the tensor product, for instance), or the arrows in the Eilenberg-Zilber theorem, can be tamed by using a symbolic representation, which allows us to model them in the frame of simplicial polynomials developed in [12].

The morphisms *AW*, *EML* and *SH* defining the Eilenberg-Zilber reduction are natural transformations between the functors $C(-) \circ (- \times -)$ and $(- \otimes -) \circ (C(-) \times C(-))$:

$$\begin{tikzcd}
\mathcal{S} \times \mathcal{S} \arrow[r, "(-\times-)"] \arrow[d, "C(-)\times C(-)"'] & \mathcal{S} \arrow[d, "C(-)"] \\
\mathcal{CC} \times \mathcal{CC} \arrow[r, "(-\otimes-)"'] & \mathcal{CC}
\end{tikzcd}$$

In the above diagram[4], $\mathcal{S}$ is the category of simplicial sets and $\mathcal{CC}$ is the category of chain complexes. Then, the functor $C(-) \circ (- \times -)$ constructs, from a pair of simplicial sets, the chain complex associated with its product (in $\mathcal{S}$), and $(- \otimes -) \circ (C(-) \times C(-))$ constructs the tensor product (in $\mathcal{CC}$) of its chain complexes. (Note that, at this description level, it is unimportant whether the chain complex is the normalized one; this situation will change at the end of the Section.)

Thus a formalization certifying the Eilenberg-Zilber algorithm requires devising a suitable representation for this kind of natural transformations.

As explained in Section 2, the tensor product in $\mathcal{CC}$ applied to the particular case of the chain complexes associated with two simplicial sets K^1 and K^2 can be expressed in each dimension n as a direct sum of $n+1$ free Abelian groups: $(C(K^1) \otimes C(K^2))_n = \bigoplus_{0 \leq i \leq n} \mathbb{Z}[K^1_{n-i} \times K^2_i]$. Furthermore, the equivalence between (finite) products and coproducts in the category $\mathcal{AG}$ of Abelian groups allows us to express the tensor product as: $(C(K^1) \otimes C(K^2))_n \cong \prod_{0 \leq i \leq n} \mathbb{Z}[K^1_{n-i} \times K^2_i]$.

In addition, the chain complex of a Cartesian product is given by: $C(K^1 \times K^2)_n = \mathbb{Z}[K^1_n \times K^2_n]$.

Therefore, using suitably both descriptions of the tensor product (as a sum when it is a source of a morphism, and as a product when it is a target of a morphism) we can conclude that, in each dimension, all the natural transformations involved in our setting can be represented as linear combinations of simpler transformations with the pattern $t : C(-,-)_{p,q} \rightarrow C(-,-)_{p',q'}$, where, for each pair (r, s), the functor $C(-,-)_{r,s} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{AG}$ is defined by $C(K^1, K^2)_{r,s} = \mathbb{Z}[K^1_r \times K^2_s]$. We should find then a way of representing morphisms between functors with the shape $C(-,-)_{r,s} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{AG}$.

[4]Be careful when reading that diagram; it is commutative only up to *homological equivalence* (this is a consequence of the Eilenberg-Zilber theorem).

An argument similar to the one presented in Section 6 of [12] (there applied to morphisms between functors $C(-)_r : \mathcal{S} \to \mathcal{AG}$), allows representing such a transformation $t : C(-,-)_{p,q} \to C(-,-)_{p',q'}$ as a linear combination with coefficients over $\mathbb{Z}$ of pairs (f^1, f^2) where each component is a simplicial operator (a consistent composite of face and degeneracy operations). Or symbolically: $t = \sum_{\alpha=1}^{n} \lambda_\alpha (f^1_\alpha, f^2_\alpha)$, with $\lambda_\alpha \in \mathbb{Z}$.

The problem is then reduced to deal with transformations which can be written by means of linear combinations of pairs of simplicial operators. These polynomials are, as it will be shown later, an extension of the framework of simplicial polynomials described in [12].

As an example, let us consider the case of differential morphisms. The differential operator in the Cartesian product can be understood as a transformation $d^\times_n : C(-,-)_{n,n} \to C(-,-)_{n-1,n-1}$. It is defined, in each dimension $n \geq 1$, by $d^\times_n = \sum_{i=0}^{n} (-1)^i \cdot (\partial_i, \partial_i)$.

As for the differential in the tensor product, it is convenient to interpret it as $d^\otimes_n : C(-,-)_{n,0} \oplus \ldots \oplus C(-,-)_{0,n} \to C(-,-)_{0,n-1} \times \ldots \times C(-,-)_{n-1,0}$. Thus, $d^\otimes_1 = (\sum_{i=0}^{1} (-1)^i (I, \partial_i) \quad \sum_{i=0}^{1} (-1)^i (\partial_i, I))$ has two components (in the previous expression, the *identity* transformation is denoted by I). And $d^\otimes_n$ has $n \times (n+1)$ components, such that the (i,j)-component, when $0 \leq i < n$ and $0 \leq j < n$, is given by:

$$d^\otimes_n(i,j) = \begin{cases} \bar{0} & \text{if } i > j \text{ or } j > i+1 \\ (-1)^i \sum_{\alpha=0}^{n-i} (-1)^\alpha (I, \partial_\alpha) & \text{if } i = j \\ \sum_{\alpha=0}^{i+1} (-1)^\alpha (\partial_\alpha, I) & \text{if } i = j-1 \end{cases} \tag{1}$$

Analogously, the maps AW_n and EML_n have $(n+1)$ components and SH_n is a single component.

Another important aspect to take into account is the normalization process. In fact, the statement of the Eilenberg-Zilber theorem describes a reduction between the *normalized* chain complexes (where elements are linear combinations of non-degenerate simplices). In the Cartesian product case, $C(K^1 \times K^2)$ consists, in each dimension n, of linear combinations of non-degenerate n-simplices from $K^1 \times K^2$; in other words, simplices which are pairs (x, y) such that they cannot be expressed as $(x, y) = \eta^\times_i(\bar{x}, \bar{y})$, for some $0 \leq i \leq n-1$ and $(\bar{x}, \bar{y}) \in K^1_{n-1} \times K^2_{n-1}$. It can be then considered as a quotient: $C(K^1 \times K^2) \cong C(\widehat{K^1 \times K^2})/D(K^1 \times K^2)$, being $D(K^1 \times K^2)$ the subcomplex generated by the combinations of degenerate simplices in $K^1 \times K^2$.

The case of the tensor product is a bit more complicated, but also admits a description in terms of Abelian groups quotients. Let us recall the complex $C(K^1) \otimes C(K^2)$, in each dimension n, is given by $(C(K^1) \otimes C(K^2))_n = \bigoplus_{0 \leq i \leq n} (C_{n-i}(K^1) \otimes C_i(K^2))$. By applying properties of free Abelian groups, we can get (here K^D_i denotes the set of degenerate i-simplices from K):

$$(C(K^1) \otimes C(K^2))_n \cong \bigoplus_{0 \leq i \leq n} (\mathbb{Z}[K^{1,ND}_{n-i}] \otimes \mathbb{Z}[K^{2,ND}_i]) \cong \bigoplus_{0 \leq i \leq n} \mathbb{Z}[K^{1,ND}_{n-i} \times K^{2,ND}_i] \cong$$

$$\bigoplus_{0 \leq i \leq n} \mathbb{Z}[(K^1_{n-i} \setminus K^{1,D}_{n-i}) \times (K^2_i \setminus K^{2,D}_i)] \cong$$

$$\bigoplus_{0 \leq i \leq n} \mathbb{Z}[K^1_{n-i} \times K^2_i]/(\mathbb{Z}[(K^1_{n-i} \times K^{2,D}_i) \cup (K^{1,D}_{n-i} \times K^2_i)]).$$

In summary, each term in the tensor product of two normalized chain complexes can be also written as a quotient of free Abelian groups. Based on the previous considerations, we say that a generator $x \otimes y$ is degenerate if some of its components (x or y)

is degenerate. Therefore, the elements of the normalized tensor product are tuples of linear combinations of pairs of non-degenerate simplices.

Therefore, both the Cartesian product and the tensor product allow us to ignore the normalization along the computing process (when dealing with morphisms), and it will be enough to apply the corresponding equality relation (pass to the quotient) at the end of the mentioned process.

3.2 Generalizing simplicial polynomials

From the discussion in the previous Section 3.1, we know that the basic components of the reduction morphisms in the Eilenberg-Zilber theorem (EZ theorem, in short) are mappings from $\mathbb{Z}[K^1_p \times K^2_q]$ to $\mathbb{Z}[K^1_{p'} \times K^2_{q'}]$. More concretely, these morphisms can be expressed as linear combinations of pairs of maps which are coherent composites 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 the next section). Nevertheless, it turns out that most of the reasoning applied to prove the EZ theorem is carried out viewing those compositions of simplicial operators (and its 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 (or pair) simplicial polynomials*; they are a representation of morphisms as symbolic expressions, built using lists and natural numbers. In this polynomial framework, we can prove, in essence, the properties stated by the EZ theorem. This approach is similar to the one presented in [12], where we used simplicial polynomials to represent morphisms from $\mathbb{Z}[K_n]$ to $\mathbb{Z}[K_m]$. Now we generalize them to represent morphisms acting on linear combinations of pairs of simplices.

Let us start with an example. Consider $\partial^6_2\eta^5_3\partial^6_2\eta^5_5\eta^4_2\partial^5_1$, 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 see the simplicial identities as rewriting rules, applied from left to right, we can write this composition in *canonical form*: $\eta_4\eta_2\partial_1\partial_3$. 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 followed by a strictly increasing sequence of faces. This canonical form is what we call a *simplicial term* and we represent it in ACL2 as a list of two lists of natural numbers, the first strictly decreasing and the second strictly increasing (in our example, `((4 2) (1 3))`). In our ACL2 formalization, the function `st-p` recognizes those ACL2 objects representing simplicial terms.

Since the morphisms we want to represent act on linear combinations of pairs of simplices, we first extend the notion of simplicial term to consider *pairs of simplicial terms*, represented in ACL2 as lists of two `st-p` objects. Linear combinations of pairs of simplicial terms are represented as lists whose elements are, in turn, lists with an integer coefficient as its first element and a pair of simplicial terms as its second element. We restrict our representation to those linear combinations in some *canonical form*: we do not allow neither zero coefficients nor different addends with the same pair of terms, and the addends are increasingly ordered with respect to a strict total ordering on pairs of terms[5]. For example, the linear combination of pairs of simplicial terms $\boldsymbol{q_1} = 3 \cdot (\eta_3\partial_1\partial_2, \eta_1\partial_0) - 2 \cdot (\eta_4\eta_2\partial_3, \partial_0\partial_1)$ is represented by the list `((3 (((3) (1 2)) ((1) (0)))) (-2 (((4 2) (3)) (() (0 1)))))`. We call *bivariate simplicial polynomials*

[5]We compare pairs of terms using the ACL2 function `lexorder`, a total ordering on ACL2 objects; but any total ordering on pairs of simplicial terms would do for our purposes.

(or *polynomials* for short) to these linear combinations in canonical form. In our ACL2 formalization, we defined the function `psp-p` to recognize those ACL2 objects representing bivariate (pair) simplicial polynomials; we will denote `psp-p`($\boldsymbol{p}$) as $\boldsymbol{p} \in \mathcal{P}^{\times}$ (in general, we will use boldface to denote polynomials).

A basic operation defined on simplicial terms is *composition.* Given two simplicial terms, its composition is the simplicial term representing its functional composition. We emphasize that the result of a composition is returned in canonical form; for example, the composition of $\eta_4\eta_1\partial_2\partial_5$ and $\eta_1\partial_2\partial_3$ returns $\eta_4\eta_1\partial_2\partial_3\partial_6$. Composition is extended componentwise to pairs of simplicial terms.

We can also define on polynomials the operations of addition, composition and scalar (integer) product, representing the corresponding operations on the functions they represent. For example, the composition of $\boldsymbol{q_1}$ above and $\boldsymbol{q_2} = 2 \cdot (\eta_2\partial_1, \eta_0\partial_1) - (\eta_4\eta_2, \partial_0\partial_1)$ 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 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 $\boldsymbol{0}$ the zero polynomial (represented by `nil` in ACL2); $\boldsymbol{id}$ the identity polynomial (that is, a polynomial with only one pair of terms and coefficient 1; each of these terms has empty lists of faces and degeneracies); and $-\boldsymbol{p}$ the scalar product of -1 and the polynomial $\boldsymbol{p}$.

We proved in ACL2 that $(\mathcal{P}^{\times}, +, \cdot)$ is a ring, with $\boldsymbol{0}$ 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:

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 [13]). Nevertheless, note that the main advantage of requiring canonical forms is that we can easily check if two given polynomials represent the same function: just check if they are syntactically equal.

An interesting point about formalizing polynomials in ACL2 is that the ring operations can be executed, and thus we can obtain the polynomial that represents any combination of compositions and sums of morphisms represented by polynomials, simply computing it in the ACL2 command interpreter. The advantages of this executability and its impact on the proof development will be commented later.

It is also noteworthy that we can recognize syntactically polynomials that, when evaluated, always return degenerate elements (in the Cartesian product or in the tensor product), regardless of the chains on which they were applied. For example, consider the pair of simplicial terms $(\eta_6\eta_3\partial_0\partial_2, \eta_5\eta_4\eta_3\partial_1)$. Then $\eta_6\eta_3 = \eta_3\eta_5$ and $\eta_5\eta_4\eta_3 = \eta_3\eta_4\eta_3$, using the simplicial identities, and therefore that pair of simplicial terms acting on a pair of simplices (x, y) is degenerate because it is equal to $\eta_3^{\times}(u, v)$ for some pair of simplices (u, v). In general, it can be proved that this will happen to every pair of simplicial terms (t_1, t_2) such that the degeneracies lists of t_1 and t_2 are not disjoint. We call such pairs *Cartesian degenerate.* We extend this property to polynomials, and consider that it is Cartesian degenerate when all its pairs hold this property. Note that every Cartesian degenerate polynomial represents a morphism that acting on a chain of the Cartesian product $C(K^1 \times K^2)$ will always return the 0 chain (recall that degenerate pairs are erased). In ACL2, we defined a function `cdpsp-p` recognizing those ACL2 objects representing Cartesian degenerate polynomials.

As for the tensor product, the situation is a little bit different. Recall from Section 3.1, that any pair of simplices in which at least one of them is degenerate, is considered degenerate in the tensor product. Then, let (t_1, t_2) be a pair of simplicial terms such that at least the degeneracies list of t_1 or the degeneracies list of t_2 is non-empty; we call such pairs of simplicial terms *tensor degenerate*. It is clear that every tensor degenerate pair of simplicial terms represents a function that applied to any pair of simplices, obtains a pair of simplices degenerate in the tensor product. Again extending this property, we say that a polynomial is tensor degenerate if all its pairs of terms are tensor degenerate. From the previous considerations, the function represented by a tensor degenerate polynomial, will always obtain linear combinations of tensor degenerate pair of simplices, and thus they will be discarded in the tensor product. In ACL2, the function `tdpsp-p` recognizes those ACL2 objects representing tensor degenerate polynomials.

3.3 The formal proof

The ring of pair (bivariate) simplicial polynomials we have just presented provides us a framework where we can prove most of the main results needed in the proof of the Eilenberg-Zilber theorem. In this Section, we define polynomials that represent the basic components of the morphisms involved in the proof of the Eilenberg-Zilber theorem, and we show the main theorems needed to stablish the reduction properties. These theorems will be expressed as properties on expressions in the ring of pair simplicial polynomials.

First, let $\boldsymbol{\partial}_i^\times$, $\boldsymbol{\partial}_i^L$, and $\boldsymbol{\partial}_i^R$ respectively denote the pairs of simplicial terms (∂_i, ∂_i), (∂_i, I) and (I, ∂_i), considered as particular cases of polynomials. 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[6]:

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 polynomial $(\partial_0, \partial_0) - (\partial_1, \partial_1) + (\partial_2, \partial_2) - (\partial_3, \partial_3)$, and in general $\boldsymbol{d}_n^\times = \sum_{i=0}^{n} (-1)^i \cdot (\partial_i, \partial_i)$. In a similar way, we define functions `left-diff` and `right-diff` returning the polynomials representing respectively the left differential and the right differential: $\boldsymbol{d}_n^L = \sum_{i=0}^{n} (-1)^i \boldsymbol{\partial}_i^L$ and $\boldsymbol{d}_n^R = \sum_{i=0}^{n} (-1)^i \boldsymbol{\partial}_i^R$.

As for the *AW* morphism, recall that it is a function from the Cartesian product to the tensor product, so according to Section 3.1, for each dimension n we have to specify $n+1$ functions, one for each of the components of the tuple of the tensor product that *AW* returns. The following function `AW-pol`(n,i) builds the corresponding polynomial (denoted as $\boldsymbol{AW}_{n,i}$) that represents the i-th component of the *AW* morphism.

Definition: $[\boldsymbol{AW}_{n,i}]$

```
AW-pol($n$,$i$) :=
    if $n \notin \mathbb{N}^+$ then $\boldsymbol{id}$
    elseif $i < n$ then AW-pol($n-1$,$i$) $\cdot$ $\boldsymbol{\partial}_n^L$
    else AW-pol($n-1$,$i$) $\cdot$ $\boldsymbol{\partial}_{n-1}^R$
```

The previous definition is a recursive version of the formula given for the *AW* morphism in Section 2 (recursion is the only way we have in ACL2 to define an iteration). For example, $\boldsymbol{AW}_{5,3}$ returns the polynomial $(\partial_4\partial_5, \partial_0\partial_1\partial_2)$, and in general $\boldsymbol{AW}_{n,i}$ returns

[6]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.

$(\partial_{i+1}\cdots\partial_n, \partial_0\cdots\partial_{i-1})$. Note that each $\boldsymbol{AW}_{n,i}$ represents a function from dimension (n,n) to dimension $(i, n-i)$.

Let us now define the polynomial counterparts of the EML morphism. Since EML is a morphism from the tensor product to the Cartesian product, according to what was explained in Section 3.1, for each dimension n, we have to specify $n+1$ components, one for each i, j such that $i+j=n$ (the result of the EML morphism on an n-dimensional tuple of the tensor product is obtained summing the respective results of each of these components). The function `EML-pol`(i,j) (denoted as $\boldsymbol{EML}_{i,j}$) builds the polynomial representing the component of the EML morphism corresponding to dimension (j,i):

Definition: [$\boldsymbol{EML}_{i,j}$]

```
EML-pol($i$,$j$) :=
    if $i \notin \mathbb{N}^+ \wedge j \notin \mathbb{N}^+$ then $\boldsymbol{id}$
    elseif $i \notin \mathbb{N}^+$ then $\boldsymbol{\eta}^R_{j-1}$ · EML-pol($i$,$j-1$)
    elseif $j \notin \mathbb{N}^+$ then $\boldsymbol{\eta}^L_{i-1}$ · EML-pol($i-1$,$j$)
    else $\boldsymbol{\eta}^L_{i+j-1}$ · EML-pol($i-1$,$j$) + $(-1)^i \cdot \boldsymbol{\eta}^R_{i+j-1}$· EML-pol($i$,$j-1$)
```

This definition is a recursive version of the formula of the EML morphism given in Section 2. For example, $\boldsymbol{EML}_{1,2}$ returns the polynomial $(\eta_0, \eta_2\eta_1) - (\eta_1, \eta_2\eta_0) + (\eta_2, \eta_1\eta_0)$, and in general $\boldsymbol{EML}_{i,j}$ has an addend for every (i,j)-shuffle. The above definition is based on how (i,j)-shuffles can be obtained recursively from $(i-1,j)$-shuffles and $(i,j-1)$ shuffles.

Finally, let us define the polynomial version of the SH morphism. In this case we have only one component for each dimension n, since it is a function from the chains of dimension n in the Cartesian product to chains of dimension $n+1$ also in the Cartesian product.

Before defining the polynomial representing SH, we have to define an operation on polynomials called *derivative* [19]. 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 increasing 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 polynomial $\boldsymbol{p}$ (we will denote it as $\boldsymbol{p}'$).

The following function `SH-pol`(n), inspired by [19], obtains the corresponding polynomial representing the SH homomorphism in dimension n[7]:

Definition: [$\boldsymbol{SH}_n$]

```
SH-pol($n$) :=
    if $n \notin \mathbb{N}^+$ then $\mathbf{0}$
    else $-1 \cdot (($SH-pol($n-1$)$)' + (\sum_{i=0}^{n} \boldsymbol{EML}_{n-i,i} \cdot \boldsymbol{AW}_{n,i})' \cdot \boldsymbol{\eta}^\times_0)$
```

Once defined all the polynomial counterparts of the morphisms that appear in the reduction version of the Eilenberg-Zilber theorem, we can establish the main properties required. These theorems are formalized as properties of expressions in the ring of pair simplicial polynomials. First, the following are the identities corresponding to the properties stating that AW and EML are chain morphisms:

Theorem: `AW-pol-chain-complex-morphism`

$n \in \mathbb{N} \wedge i \in \mathbb{N} \wedge i < n$
$\quad \to \boldsymbol{AW}_{n-1,i} \cdot \boldsymbol{d}^\times_n = \boldsymbol{d}^L_{i+1} \cdot \boldsymbol{AW}_{n,i+1} + (-1)^i \cdot \boldsymbol{d}^R_{n-i} \cdot \boldsymbol{AW}_{n,i}$

Theorem: `EML-pol-chain-complex-morphism`

[7]The summatory in the definition is defined recursively by an auxiliary function.

$i \in \mathbb{N}^+ \land j \in \mathbb{N}^+$
$\quad \to \boldsymbol{d}^{\times}_{i+j} \cdot \boldsymbol{EML}_{i,j} = \boldsymbol{EML}_{i,j-1} \cdot \boldsymbol{d}^{L}_{j} + (-1)^j \cdot \boldsymbol{EML}_{i-1,j} \cdot \boldsymbol{d}^{R}_{i}$

Second, the following ACL2 theorems establish properties (1) to (5) required in the definition of reduction (Definition 3), for $\boldsymbol{AW}$, $\boldsymbol{EML}$ and $\boldsymbol{SH}$. To understand the statement of these properties, recall that, as commented at the end of Section 3.2, if a polynomial $\boldsymbol{p}$ is cartesian degenerate (that is, if `cdpsp-p`($\boldsymbol{p}$)), then the function that the polynomial represents would always return the 0 chain of the Cartesian product; as a particular case, if `cdpsp-p`($\boldsymbol{q} - \boldsymbol{id}$), then the function represented by $\boldsymbol{q}$ is the identity function on the cartesian product. Analogously for polynomials representing functions that return tuple components of the tensor product, but in this case, we need the tensor degenerate property `tdpsp-p`.

Theorem (1): `tdpsp-AW-pol-EML-pol-id`
$\quad i \in \mathbb{N} \land j \in \mathbb{N} \to$ `tdpsp-p`$(\boldsymbol{AW}_{i+j,j} \cdot \boldsymbol{EML}_{i,j} - \boldsymbol{id})$

Theorem (1): `tdpsp-AW-pol-EML-pol`
$\quad i \in \mathbb{N} \land j \in \mathbb{N} \land k \in \mathbb{N} \land j \neq k \land k \leq i + j \to$ `tdpsp-p`$(\boldsymbol{AW}_{i+j,k} \cdot \boldsymbol{EML}_{i,j})$

Theorem (2): `cdpsp-diff-SH-pol-SH-pol-diff-EML-AW-id`
$\quad n \in \mathbb{N}^+$
$\quad\quad \to$ `cdpsp-p`$(\boldsymbol{d}^{\times}_{n+1} \cdot \boldsymbol{SH}_n + \boldsymbol{SH}_{n-1} \cdot \boldsymbol{d}^{\times}_{n} + \sum_{i=0}^{n} \boldsymbol{EML}_{n-i,i} \cdot \boldsymbol{AW}_{n,i} - \boldsymbol{id})$

Theorem (3): `tdpsp-AW-pol-SH-pol`
$\quad n \in \mathbb{N} \land p \in \mathbb{N} \land p \leq n + 1 \to$ `tdpsp-p`$(\boldsymbol{AW}_{n+1,p} \cdot \boldsymbol{SH}_n)$

Theorem (4): `cdpsp-SH-pol-EML-pol`
$\quad i \in \mathbb{N} \land j \in \mathbb{N} \to$ `cdpsp-p`$(\boldsymbol{SH}_{i+j} \cdot \boldsymbol{EML}_{i,j})$

Theorem (5): `cdpsp-SH-pol-SH-pol`
$\quad n \in \mathbb{N} \to$ `cdpsp-p`$(\boldsymbol{SH}_{n+1} \cdot \boldsymbol{SH}_n)$

The formalization of property (1) by the two first ACL2 theorems above needs some explanation. They establish, in the polynomial setting, that the composition of $\boldsymbol{AW}_{n,j}$ with each of the components $\boldsymbol{EML}_{n-k,k}$ $(0 \leq j, k \leq n)$ returns, in the tensor product, the zero chain, except for $k = j$, which is the identity. Note that when applying the composition of AW_n and EML_n to a tuple of the tensor product, each component of the result is a sum obtained by composing the corresponding component of AW_n with the sum of the respective applications of each component of EML_n to the components of the tuple. Combining the two first theorems, we have that in that sum, only one addend is the identity and the rest are null. Thus, the composition of AW_n and EML_n is the identity in the tensor product.

The ACL2 theorems presented in this section establish the Eilenberg Zilber theorem, expressed in our polynomial framework. These properties are non trivial but, roughly speaking, they all are proved by induction on natural numbers and applying algebraic properties in the ring of pair simplicial polynomials. In the next section, we will explain in some detail the proof of one of these properties.

3.4 A detailed example: the main lemma in a reduction

We illustrate the kind of reasoning carried out, presenting a sketch of the proof of the property `cdpsp-diff-SH-pol-SH-pol-diff-EML-AW-id` (property (2) of reductions).

That is, we will prove that $\boldsymbol{d}^{\times}_{n+1} \cdot \boldsymbol{SH}_n + \boldsymbol{SH}_{n-1} \cdot \boldsymbol{d}^{\times}_n + \sum_{i=0}^{n} \boldsymbol{EML}_{n-i,i} \cdot \boldsymbol{AW}_{n,i} - \boldsymbol{id}$ is a cartesian degenerate polynomial, for every $n \geq 1$. In particular, we will prove that this expression is equal to the cartesian degenerate polynomial $-\boldsymbol{\eta}^{\times}_{n-1} \cdot \boldsymbol{\partial}^{\times}_n$. It is worth pointing out that we first conjectured the general expression of this cartesian degenerate polynomial by computing the polynomial expression in ACL2, for several values of n. This is possible due to the executability of the polynomial ring operations in ACL2.

In the following, let $\boldsymbol{E}_n\boldsymbol{A}_n$ be an abbreviation to denote the polynomial $\sum_{i=0}^{n} \boldsymbol{EML}_{n-i,i} \cdot \boldsymbol{AW}_{n,i}$. Precisely, what we prove by induction on the natural numbers is the following equivalent property:

$$\boldsymbol{d}^{\times}_{n+1} \cdot \boldsymbol{SH}_n = -\boldsymbol{SH}_{n-1} \cdot \boldsymbol{d}^{\times}_n - \boldsymbol{E}_n\boldsymbol{A}_n + \boldsymbol{id} - \boldsymbol{\eta}^{\times}_{n-1} \cdot \boldsymbol{\partial}^{\times}_n$$

For $n = 1$, we can compute the expressions: $\boldsymbol{E}_1\boldsymbol{A}_1 = (\eta_0\partial_1, I) + (I, \eta_0\partial_0)$, $\boldsymbol{d}^{\times}_2 \cdot \boldsymbol{SH}_1 = \boldsymbol{id} - \boldsymbol{\eta}^{\times}_0 \cdot \boldsymbol{\partial}^{\times}_1 - (\eta_0\partial_1, I) - (I, \eta_0\partial_0)$ and $\boldsymbol{SH}_0 \cdot \boldsymbol{d}^{\times}_1 = \boldsymbol{0}$. Therefore, the property holds in this case.

Let us now prove the property for $n > 1$, assuming the property for $n - 1$. For that we will need a number of lemmas, that we list here, omitting their proof:

(a) The derivative operation distributes over addition, composition and scalar product of polynomials.

(b) $\boldsymbol{d}^{\times}_{n+1} = \boldsymbol{\partial}^{\times}_0 - (\boldsymbol{d}^{\times}_n)'$

(c) $\boldsymbol{\partial}^{\times}_0 \cdot (\boldsymbol{p})' = \boldsymbol{p} \cdot \boldsymbol{\partial}^{\times}_0$, for every polynomial $\boldsymbol{p}$

(d) $\boldsymbol{d}^{\times}_n \cdot \boldsymbol{E}_n\boldsymbol{A}_n = \boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1} \cdot \boldsymbol{d}^{\times}_n$ (that is, $\boldsymbol{E}_n\boldsymbol{A}_n$ represents a differential morphism).

(e) $(\boldsymbol{d}^{\times}_n)' \cdot \boldsymbol{\eta}^{\times}_0 + \boldsymbol{\eta}^{\times}_0 \cdot \boldsymbol{\partial}^{\times}_0 - \boldsymbol{id} = \boldsymbol{\eta}^{\times}_0 \cdot \boldsymbol{d}^{\times}_n$

To prove the intended property, we will see that the expressions $\boldsymbol{d}^{\times}_{n+1} \cdot \boldsymbol{SH}_n$ and $-\boldsymbol{SH}_{n-1} \cdot \boldsymbol{d}^{\times}_n - \boldsymbol{E}_n\boldsymbol{A}_n + \boldsymbol{id} - \boldsymbol{\eta}^{\times}_{n-1} \cdot \boldsymbol{\partial}^{\times}_n$ are equal, rewriting both to a common expression.

- Rewriting $\boldsymbol{d}^{\times}_{n+1} \cdot \boldsymbol{SH}_n$:

  If we apply lemma (b) and the definition of $\boldsymbol{SH}_n$, and then apply lemma (a), we have:

$$-\boldsymbol{\partial}^{\times}_0 \cdot (\boldsymbol{SH}_{n-1})' - \boldsymbol{\partial}^{\times}_0 \cdot (\boldsymbol{E}_n\boldsymbol{A}_n)' \cdot \boldsymbol{\eta}^{\times}_0 + (\boldsymbol{d}^{\times}_n \cdot \boldsymbol{SH}_{n-1})' + (\boldsymbol{d}^{\times}_n \cdot \boldsymbol{E}_n\boldsymbol{A}_n)' \cdot \boldsymbol{\eta}^{\times}_0$$

  We now apply lemma (c) (twice), lemma (d) and the equality $\boldsymbol{\partial}^{\times}_0 \cdot \boldsymbol{\eta}^{\times}_0 = \boldsymbol{id}$ (which is a direct consequence of the fifth simplicial identity), rewriting the expression to:

$$-\boldsymbol{SH}_{n-1} \cdot \boldsymbol{\partial}^{\times}_0 - \boldsymbol{E}_n\boldsymbol{A}_n + (\boldsymbol{d}^{\times}_n \cdot \boldsymbol{SH}_{n-1})' + (\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1} \cdot \boldsymbol{d}^{\times}_n)' \cdot \boldsymbol{\eta}^{\times}_0$$

  We replace the first occurrence of $\boldsymbol{SH}_{n-1}$ by its definition, we also apply the induction hypothesis to $\boldsymbol{d}^{\times}_n \cdot \boldsymbol{SH}_{n-1}$ and distribute the derivative over addition, and we also distribute in some addends the derivative over composition, obtaining:

$$(\boldsymbol{SH}_{n-2})' \cdot \boldsymbol{\partial}^{\times}_0 + (\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' \cdot \boldsymbol{\eta}^{\times}_0 \cdot \boldsymbol{\partial}^{\times}_0 - \boldsymbol{E}_n\boldsymbol{A}_n +$$

$$-(\boldsymbol{SH}_{n-2})' \cdot (\boldsymbol{d}^{\times}_{n-1})' - (\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' + \boldsymbol{id} - \boldsymbol{\eta}^{\times}_{n-1} \cdot \boldsymbol{\partial}^{\times}_n + (\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' \cdot (\boldsymbol{d}^{\times}_n)' \cdot \boldsymbol{\eta}^{\times}_0$$

  Factoring out $(\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})'$ and applying lemma (e), we finally have:

$$(\boldsymbol{SH}_{n-2})' \cdot \boldsymbol{\partial}^{\times}_0 - \boldsymbol{E}_n\boldsymbol{A}_n - (\boldsymbol{SH}_{n-2})' \cdot (\boldsymbol{d}^{\times}_{n-1})' + \boldsymbol{id} - \boldsymbol{\eta}^{\times}_{n-1} \cdot \boldsymbol{\partial}^{\times}_n +$$

$$(\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' \cdot \boldsymbol{\eta}^{\times}_0 \cdot \boldsymbol{d}^{\times}_n$$

- Rewriting $-\boldsymbol{SH}_{n-1} \cdot \boldsymbol{d}_n^\times - \boldsymbol{E}_n\boldsymbol{A}_n + \boldsymbol{id} - \boldsymbol{\eta}_{n-1}^\times \cdot \boldsymbol{\partial}_n^\times$:

  Expanding the definition of $\boldsymbol{SH}_{n-1}$ and distributing composition over addition, we have:

$$(\boldsymbol{SH}_{n-2})' \cdot \boldsymbol{d}_n^\times + (\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' \cdot \boldsymbol{\eta}_0^\times \cdot \boldsymbol{d}_n^\times - \boldsymbol{E}_n\boldsymbol{A}_n + \boldsymbol{id} - \boldsymbol{\eta}_{n-1}^\times \cdot \boldsymbol{\partial}_n^\times$$

  And applying lemma (b) to the first occurrence of $\boldsymbol{d}_n^\times$, and distributivity we have:

$$(\boldsymbol{SH}_{n-2})' \cdot \boldsymbol{\partial}_0^\times - (\boldsymbol{SH}_{n-2})' \cdot (\boldsymbol{d}_{n-1}^\times)' +$$

$$(\boldsymbol{E}_{n-1}\boldsymbol{A}_{n-1})' \cdot \boldsymbol{\eta}_0^\times \cdot \boldsymbol{d}_n^\times - \boldsymbol{E}_n\boldsymbol{A}_n + \boldsymbol{id} - \boldsymbol{\eta}_{n-1}^\times \cdot \boldsymbol{\partial}_n^\times$$

It is now clear (applying commutativity of addition of polynomials) that both expressions rewrite to the same expression. Therefore they are equal and we have the property proved. Note that the proof is done only applying induction and symbolic rewriting using definitions, previously proved lemmas and the ring properties of simplicial polynomials.

3.5 Main simplification strategies

As we have just seen in Section 3.4, the proof of the polynomial properties needed to establish the Eilenberg-Zilber theorem uses a great amount of simplification mechanisms, crucial for a successful mechanical proof. These simplification mechanisms are programmed as rewriting rules. In our formalization, these rules are essentially of four types: (a) rules stating that under certain conditions, a polynomial expression has properties `cdpsp-p` or `tdpsp-p`; (b) rules stating properties derived from the simplicial identities (as for example, some of the lemmas used in Section 3.4); (c) ring properties of the ring of polynomials; and (d) meta-rules, implementing algebraic rewriting that cannot be expressed as single rewriting rules.

Let us comment more on this last type of rule, illustrating how we used meta rules, by an example. Consider the following situation: if we previously prove $x = y$, then we could conclude $x \cdot z = y \cdot z$. Of course, this can be formalized by means of a rewriting rule; however there are many situations where this rule is not applicable although an analogous simplification could be done; for example $x_1 \cdot z = y_1 \cdot z + y_2 \cdot z$ when $x_1 = y_1 + y_2$ or $x_1 \cdot z + x_2 \cdot z = y_1 \cdot z + y_2 \cdot z$ when $x_1 + x_2 = y_1 + y_2$.

In these cases we could think in a rule that would extract the right common factor, then applying the previous rule. But "extract the right common factor" cannot be expressed as a single rewriting rule, since it should manage as many situations as addends could appear in an expression, and that cannot be expressed as a single pattern. Of course we could define as many rewriting rules as needed in the formalization, but this is expensive in terms of user's labor to develop the set of rules and of system's labor in sorting through such a number of rules.

In situations like the one described, ACL2 *meta functions* [9] are a convenient tool. A meta function is a piece of code that performs a syntactic transformation on some terms. When this transformation is proved to be semantically correct by means of a *meta rule*, the meta function extends the operations of the ACL2's simplifier.

We have defined several meta functions to cope with usual simplification processes in the general context of polynomials. Continuing with the example, let us consider a generic associative operator $+$ with identity (for example, addition of polynomials) and a generic associative operator $\cdot$ with identity (for example, composition of polynomials) and right-distributive over $+$. The *right common factor meta rule* has two cases depending on whether the common factor has an inverse element or not. Given an expression of the form $x_1 \cdot z + \ldots + x_n \cdot z = y_1 \cdot z + \ldots + y_m \cdot z$, this rule extracts the common factor z and, if there exists z' such that $z \cdot z' = 1$, the expression is replaced with the equivalent one $x_1 + \ldots + x_n = y_1 + \ldots + y_m$. Otherwise, the expression is reduced to true, provided it can be proved that $x_1 + \ldots + x_n = y_1 + \ldots + y_m$. In a similar way we have defined a *left common factor meta rule*.

More meta rules has been introduced, implementing several useful algebraic simplification mechanisms in presence of associativity, commutativity and distributivity of the operators. These rules have been proved correct in a generic setting, for generic operators having those properties. Then, they have been instantiated to the particular case of simplicial polynomials. See details in [13].

4 Proof computational content and experimental aspects

This section is devoted to explore the computational content of the proof, handling it to get a statement of the theorem near the standard mathematical presentation.

Being ACL2 also a programming language, it is clear that the functions defined to implement the proof are executable. That is, we can produce simplicial polynomials representing the EZ morphisms, for each dimension. Nevertheless, this formalization is missing the functional interpretation of polynomials: we should be able to apply and execute them on concrete chains of simplices of concrete simplicial sets. For that, we need a presentation of the EZ theorem where the notions of simplicial sets, chains, cartesian product and tensor product are made explicit, and where the morphisms are defined as ACL2 functions. This is what we call the *functional* (or *standard*) formalization of the theorem, and we will present it in Section 4.1. As we will see, the main theorems in this standard formalization are obtained translating from the corresponding theorems of the polynomial formalization of the previous section.

Once proved, the functional or standard formalization of the EZ theorem can be instantiated on concrete simplicial sets, where we can execute the morphisms. In Section 4.2, it is particularized on Δ, the *standard simplex*. In this simplicial set the simplicial equalities are the only constraints, and so any simplicial formula is expressed generically on it. Then, in Section 4.3, the results obtained with our certified ACL2 programs are compared with the actual *Kenzo* results.

4.1 Operational interpretation of the proof

4.1.1 Formalizing simplicial sets and chain complexes

We represent a simplicial set and its associated chain complex in the same way as we did it in the formalization of the Normalization Theorem [12]. We now give a brief overview to the main ideas.

Let us first deal with how we represent simplicial sets. Note that a simplicial set is characterized by a set K and a family of functions (faces and degeneracies) having certain properties. Since the EZ 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 [10].

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 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`
$(x \in K_m \wedge m \in \mathbb{N}^+ \wedge i \in \mathbb{N} \wedge i \leq m) \to \partial_i^m(x) \in K_{m-1}$

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

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.

We can now formalize the notions of degenerate and non-degenerate simplices. First, the predicate `Kd`(n,x) defines the property of being a degenerate n-simplex:

Definition: $[x \in K_n^D]$
`Kd`(n,x) $:= \exists y,i \; (i \in \mathbb{N} \wedge i < n \wedge y \in K_{n-1} \wedge \eta_i^{n-1}(y) = x)$

The existential quantification in the definition is introduced in ACL2 using the `defun-sk` construct (this construct is the way in which ACL2 offers (limited) support for existential quantification). Having defined degenerate simplices, non-degenerate simplices can be easily defined:

Definition: $[x \in K_n^{ND}]$
`Kn`(n,x) $:= x \in K_n \wedge x \notin K_n^D$

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: 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 function `ssn-p` which recognizes two-element lists whose elements are a non-null integer and a non-degenerate simplex; it also uses the auxiliary function `ssn-<` which defines a strict ordering 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. In this paper, we will denote these operations, respectively, as $c_1 + c_2$ and $k \cdot c$ (omitting the dimension, for the sake of readability)[8]. These operations act on chains in the canonical form described above, and return chains also in canonical form. We proved that the set

[8]We are overloading the symbols, using the same notation for the operation on chains and on polynomials, but the distinction will be clear from the context.

of chains of a given dimension is an Abelian group with respect to addition, where the identity is represented by the empty list (denoted here as 0).

To complete the definition of the chain complex associated to a simplicial set, we need to define the differential homomorphism and prove that the boundary condition holds. First, face and degeneracy maps are defined to act on chains, easily extending them by linearity; in this paper, we will use the same notation (that is, ∂_i^n and η_i^n) regardless whether these operations are acting on simplices or on chains. Now, as we said in Section 2, the differential of a chain $c \in C_n(K)$ is defined as $d_n(c) = \sum_{i=0}^{n}(-1)^i \partial_i^n(c)$, but taking into account that in the resulting chain, any degenerate addend has to be erased. The following functions implement it:

Definition:
```
F-norm($n$,$c$) :=
    **if** endp($c$) **then** 0
    **elseif** ssn-p($n$,first($c$))
      **then** first($c$) + F-norm($n$,rest($c$))
    **else** F-norm($n$,rest($c$))
```

Definition:
```
diff-aux($n$,$i$,$c$) :=
    **if** $i \notin \mathbb{N}^+$ **then** $\partial_0^n(c)$
    **else** $(-1)^i \cdot \partial_i^n(c)$ + diff-aux($n$,$i-1$,$c$)
```

Definition: $[d_n(c)]$
```
diff($n$,$c$) := F-norm($n-1$,diff-aux($n$,$n$,$c$))
```

The function `F-norm` above takes a linear combination of simplices, in which there are possibly some degenerate addends and returns the chain with those addends erased. The function `diff`(n,c), denoted as $d_n(c)$, defines the differential homomorphism. Note that it uses an auxiliary recursive function `diff-aux`.

We prove the boundary condition for the differential function just defined, completing the formalization of the chain complex associated to a simplicial set. The following theorem establishes it:

Theorem: `diff-diff-null`
$n \in \mathbb{N}^+ \wedge c \in C_{n+1}(K) \rightarrow d_n(d_{n+1}(c)) = 0$

The Eilenberg-Zilber theorem establishes a result about any two simplicial sets and the relation between their Cartesian and tensor products. This means that, to formally state the premises of the theorem in ACL2, we have to define two generic simplicial sets (say K^1 and K^2) and their associated chain complexes. Thus we introduce functions `K1`, `d1` and `n1` (by means of the encapsulation principle), assuming the corresponding simplicial identities, and the same for `K2`, `d2` and `n2`. Then we replay all the definitions and theorems needed to formalize the respective chain complexes, in an analogous way as we have just shown. That is, we define for K^1 the predicates `K1n` and `sc1n` (respectively recognizing non-degenerate simplices and chains in $C(K^1)$), the differential function `diff1` and prove the corresponding boundary condition, among other useful lemmas. We do it for K^2 in an analogous way. In the following, we will denote as d_n^1 and d_n^2 the corresponding differentials for $C(K^1)$ and $C(K^2)$. For the sake of readability, we will denote in the same way (∂_i^n and η_i^n) the faces and degeneracies of both simplicial sets, although it has to be clear that in the formalization they are different families of functions.

A technical comment is worth pointing out here. In principle, to obtain the theories (definitions and theorems) corresponding to the chain complexes $C(K^1)$ and $C(K^2)$, we would have to duplicate the proof effort carried out. Fortunately, a rule of inference in ACL2, called *functional instantiation*, allows us to infer theorems that can be obtained instantiating the function symbols of a previously proved theorem, replacing them with other function symbols, provided it can be proved that the new functions satisfy the constraints assumed on the replaced functions. So, we can define one generic simplicial set and its associated chain complex, and obtain other generic simplicial sets by functional instantiation. Moreover, in our case, this instantiation is done in a completely automatic way: although ACL2 offers no native support for functionally instantiate a whole theory (that is, a collection of definitions and theorems about them), we used a tool called `definstance`, that allows us to automatically generate functional instantiations of a theory, simply giving the corresponding "names substitution" (see [14] for details on the `definstance` tool).

4.1.2 Cartesian product of simplicial sets

The Cartesian product of the simplicial sets K^1 and K^2 is easily defined. First, we define the functions `Kx2`(p,q,x) recognizing pairs $x \in K^1_p \times K^2_q$; as a particular case, we define the function `Kx` formalizing $(K^1 \times K^2)_n$:

Definition: $[x \in K^1_p \times K^2_q]$
$\texttt{Kx2}(p,q,x) := \texttt{consp}(x) \wedge \texttt{first}(x) \in K^1_p \wedge$
$\qquad \texttt{consp}(\texttt{rest}(x)) \wedge \texttt{second}(x) \in K^2_q \wedge \texttt{rest}(\texttt{rest}(x)) = ()$

Definition: $[x \in (K^1 \times K^2)_n]$
$\texttt{Kx}(n,x) := \texttt{Kx2}(n,n,x)$

The face and degeneracy operators for the Cartesian product of $K^1 \times K^2$ are defined componentwise from the corresponding operators of K^1 and K^2:

Definition: $[\partial_i^{\times,n}(x)]$
$\texttt{dx}(n,i,x) := (\partial_i^n(\texttt{first}(x)), \partial_i^n(\texttt{second}(x)))$

Definition: $[\eta_i^{\times,n}(x)]$
$\texttt{nx}(n,i,x) := (\eta_i^n(\texttt{first}(x)), \eta_i^n(\texttt{second}(x)))$

It is straightforward to prove that $\partial_i^{\times,n}(x)$ and $\eta_i^{\times,n}(x)$ hold the simplicial identities, and thus we can define, by functional instantiation (and again in an automatic way using `definstance`), all the definitions and theorems corresponding to the associated complex chain. In particular, we define a recognizer for chains of $C_n(K^1 \times K^2)$ (function `SCxn-p`(n,x)), and a function `Fx-norm`(n,c) that erases the degenerate addends (w.r.t. the cartesian product) of linear combinations of pairs of n-simplices. Also we define the differential homomorphism $d_n^\times$ (function `Cx-diff`(n,c)) and prove the corresponding boundary condition, thus completing the formalization of the Cartesian product $C(K^1 \times K^2)$.

4.1.3 Tensor product

Since the tensor product of two simplicial chain complexes cannot be obtained as the chain complex associated to a simplicial set, in order to formalize it in ACL2, we cannot use the same technique used to define the Cartesian product. In this case, we have to directly define the sets $(C(K^1) \otimes C(K^2))_n$ and the corresponding differential homomorphism.

As discussed in Section 3.1, the elements of $(C(K^1) \otimes C(K^2))_n$ can be identified with $(n+1)$-tuples (lists in our case) $(c_0, c_1, \dots, c_n)$, where for each $0 \leq i \leq n$, $c_i \in \mathbb{Z}[K^{1,ND}_{n-i} \times K^{2,ND}_{i}]$. The following function `Cn+` formalizes this, from the auxiliary recursive function `Cn+-seq`, which deals with the iteration (here, the function `SCx2n-p`(p, q, c) recognizes linear combinations in $\mathbb{Z}[K^{1,ND}_p \times K^{2,ND}_q]$):

Definition:
```
Cn+-seq($n$,$p$,$l$) :=
    **if** endp($l$) **then nil**
    **elseif** $p \notin \mathbb{N}^+$ **then** SCx2n-p(0,$n$,first($l$)) $\wedge$ rest($l$) = **nil**
    **else** SCx2n-p($p$,$n-p$,first($l$)) $\wedge$ Cn+-seq($n$,$p-1$,rest($l$))
```

Definition: $[c \in (C(K^1) \otimes C(K^2))_n]$
```
Cn+($n$,$c$) := Cn+-seq($n$,$n$,$c$)
```

To define the differential homomorphism in the tensor product, first we introduce some notation. Let d^L_p and d^R_q denote the functions defined on $K^{1,ND}_p \times K^{2,ND}_q$ such that $d^L_p(x,y) = (d^1_p(x), y)$ and $d^R_q(x,y) = (x, d^2_q(y))$. Now, for every generator $(x,y) \in K^{1,ND}_p \times K^{2,ND}_q$ (with $p+q=n$), the differential in the tensor product (see Definition 5) can be written as: $d^{\otimes}_n(x,y) = d^L_p(x,y) + (-1)^p d^R_q(x,y)$. As usual, d^L_p, d^R_q and $d^{\otimes}_n$ are extended by linearity to $\mathbb{Z}[K^{1,ND}_p \times K^{2,ND}_q]$.

Let $c = (c_0, c_1, \dots, c_n) \in (C(K^1) \otimes C(K^2))_n$ and $e = (e_0, \dots, e_{n-1}) \in (C(K^1) \otimes C(K^2))_{n-1}$, such that $d^{\otimes}_n(c) = e$. Then from the above considerations we have that $e_j = d^L_{n-j}(c_j) + (-1)^{n-j-1} d^R_{j+1}(c_{j+1})$ (for all $0 \leq j \leq n-1$). Our definition of $d^{\otimes}_n$ is based on this last formula:

Definition:
```
diff+-seq($n$,$p$,$l$) :=
    **if** $p \notin \mathbb{N}^+$ **then nil**
    **else** cons($d^L_p$(first($l$)) + $(-1)^{p-1} \cdot d^R_{n-p+1}$(second($l$)),
               diff+-seq($n$,$p-1$,rest($l$)))
```

Definition: $[d^{\otimes}_n(c)]$
```
diff+($n$,$c$) := F+-norm($n-1$,diff+-seq($n$,$n$,$c$))
```

Note the normalization applied in the definition of `diff+`. This is needed because in our ACL2 formalization, d^L_p and d^R_q are defined applying the corresponding differential, but without erasing the degenerate addends (as in the function `diff-aux` above). Therefore, in each component of the final result obtained by `diff+-seq`(n,n,c) we have to delete all the addends corresponding to degenerate pair of simplices in the tensor product. This is precisely what the function `F+-norm` does.

From the corresponding boundary conditions of d^1_n and d^2_n, we prove the boundary condition for $d^{\otimes}_n$, as established by the following theorem (note that zero in the tensor product, denoted as $0^{\otimes}$, is the tuple with all its components equal to the zero chain):

Theorem: `diff+-diff+-null`
$n \in \mathbb{N}^+ \wedge l \in (C(K^1) \otimes C(K^2))_{n+1} \rightarrow d^{\otimes}_n(d^{\otimes}_{n+1}(l)) = 0^{\otimes}$

This completes the formalization of the tensor product $C(K^1) \otimes C(K^2)$. In the following, we explain the formalization and proof of the Eilenberg-Zilber theorem in this standard framework. For that, we have first to define the morphisms AW, EML and SH. Not surprisingly, this will be done using the corresponding polynomial versions.

4.1.4 Evaluation of bivariate simplicial polynomials

Before defining the morphisms involved in the EZ theorem, 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. This is done in a similar way to what was presented in [12].

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, 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 call 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 *valid for dimension* (m_1, m_2) if all its terms are valid for that dimension, and we say that it is *uniform* if all its terms have the same degree. If a polynomial is uniform and valid for a dimension we say that it is *well-formed* for that dimension and its degree is the common degree of its terms.

Well-formed polynomials for dimension (m_1, m_2) represent morphisms whose evaluation can be defined on $\mathbb{Z}[K^1_{m_1} \times K^2_{m_2}]$. We defined in ACL2 a function `eval-psp`($\boldsymbol{p}$,m_1,m_2,c) that computes the result of evaluating $\boldsymbol{p}$ on a linear combination c of pairs of simplices of dimension (m_1, m_2). 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 [12] for simple simplicial polynomials. See [13] for details.

4.1.5 Defining AW, EML and SH

We now define the morphisms that form the reduction in the Eilenberg-Zilber theorem, as combinations of evaluations of the polynomials defined in Section 3.3. The way we combine these evaluations depend on their intended domain and range (the Cartesian product or the tensor product).

Let us first start with the AW morphism. Recall that for each dimension n, AW_n defines a function from $C_n(K^1 \times K^2)$ to $(C(K^1) \otimes C(K^2))_n$. First we define the function `AW-aux`(p,q,n,i,c) as the evaluation of the polynomial $\boldsymbol{AW}_{n,i}$ on a linear combination c of pairs of simplices of dimension (p, q). We can prove that $\boldsymbol{AW}_{n,i}$ is a well-formed polynomial for dimension (n, n), with degree $(i - n, -i)$, so it makes sense to define `AW` on $C_n(K^1 \times K^2)$ as the result of iteratively apply `AW-aux`(n,n,n,i,c) (for $0 \leq i \leq n$) and collect each result in a tuple; in each component of this tuple we finally erase (using `F+-norm`) the possible degenerate addends (in the tensor product) that could appear. As usual, note the auxiliary recursive function `AW-seq` implementing the iteration.

Definition:

```
AW-aux($p$,$q$,$n$,$i$,$c$) := eval-psp($\boldsymbol{AW}_{n,i}$,$p$,$q$,$c$)
```

Definition:

```
AW-seq($n$,$p$,$c$) :=
    if $p \notin \mathbb{N}^+$ then list(AW-aux($n$,$n$,$n$,0,$c$))
    else cons(AW-aux($n$,$n$,$n$,$p$,$c$),AW-seq($n$,$p-1$,$c$))
```

Definition: $[AW_n(c)]$

```
AW($n$,$c$) := F+-norm($n$,AW-seq($n$,$n$,$c$))
```

We now define the EML morphism. In this case, EML_n is defined on elements of $(C(K^1) \otimes C(K^2))_n$, returning its result in $C_n(K^1 \times K^2)$. Recall that, as discussed in Section 3.1, the elements of $(C(K^1) \otimes C(K^2))_n$ are tuples of linear combinations in $\mathbb{Z}[K_p^{1,ND} \times K_q^{2,ND}]$, where we have a component for each dimension (p, q) such that $p + q = n$.

Taking this into account, we first define `EML-aux`(p,q,i,j,c) as the evaluation of the polynomial $\boldsymbol{EML}_{i,j}$ on a linear combination $c \in \mathbb{Z}[K_p^{1,ND} \times K_q^{2,ND}]$. It can be proved that $\boldsymbol{EML}_{i,j}$ is well-formed for dimension (j, i) and its degree is (i, j). Therefore it is valid to define EML_n on a tuple l as the result of iteratively applying `EML-aux`(i,$n-i$,$n-i$,i,l_i) (where $0 \leq i \leq n$) and sum each result to obtain a single chain in dimension (n, n); as with AW, we finally erase (using `Fx-norm`) the possible degenerate addends, now in the Cartesian product. Again, we need an auxiliary recursive function `EML-seq` implementing the iteration.

Definition:

```
EML-aux($p$,$q$,$i$,$j$,$c$) := eval-psp($\boldsymbol{EML}_{i,j}$,$p$,$q$,$c$)
```

Definition:

```
EML-seq($n$,$p$,$l$) :=
    if $p \notin \mathbb{N}^+$ then EML-aux(0,$n$,$n$,0,first($l$))
    else EML-aux($p$,$n-p$,$n-p$,$p$,first($l$)) + EML-seq($n$,$p-1$,rest($l$))
```

Definition: $[EML_n(c)]$

```
EML($n$,$c$) := Fx-norm($n$,EML-seq($n$,$n$,$c$))
```

Finally the definition of the SH function from the corresponding polynomial is simpler, since we do not have to deal with tuple components (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 $\boldsymbol{SH}_n$ 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 Cartesian product:

Definition: $[SH_n(c)]$

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

4.1.6 The main properties

In Section 3.3, we established, in the polynomial framework, the main properties showing that $\boldsymbol{AW}$, $\boldsymbol{EML}$ and $\boldsymbol{SH}$ form a reduction from $C(K^1 \times K^2)$ to $C(K^1) \otimes C(K^2)$. We are almost ready to translate those properties to this more standard presentation of the theorem. Before that, recall that some of those properties stated that some operations on polynomials returned tensor degenerate (`tdpsp-p`) or cartesian degenerate (`cdpsp-p`) polynomials. As we anticipated, those properties have a direct translation in this standard framework:

Theorem: `Fx-norm-eval-psp-cdpsp`
$n \in \mathbb{N}^+ \wedge i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge$ `SCx2-p`$(i,j,c) \wedge$ `cdpsp-p`$(\boldsymbol{p}) \wedge$ `uniform-psp`$(\boldsymbol{p}) \wedge$
`valid-psp`$(\boldsymbol{p},i,j) \wedge$ `degree-psp`$(\boldsymbol{p}) = (n-i, n-j)$
$\to$ `Fx-norm`$(n,$`eval-psp`$(\boldsymbol{p},i,j,c)) = 0^{\times}$

Theorem: `Fx2-norm-eval-psp-tdpsp`
$i \in \mathbb{N} \wedge j \in \mathbb{N} \wedge$ `SCx2-p`$(i,j,c) \wedge$ `tdpsp-p`$(\boldsymbol{p}) \wedge$ `uniform-psp`$(\boldsymbol{p}) \wedge$
`valid-psp`$(\boldsymbol{p},i,j) \wedge$ `degree-psp`$(\boldsymbol{p}) = (k, l)$
$\to$ `Fx2-norm`$(i+k,j+l,$`eval-psp`$(\boldsymbol{p},i,j,c)) = 0^{\times}$

That is, if we evaluate a cartesian degenerate polynomial on a linear combination of pair of simplices (under the corresponding well-formedness condition) and then we erase the addends that are degenerate in the Cartesian product, we obtain the zero chain $0^{\times}$. And an analogous result is also obtained for tensor degenerate polynomials in the tensor product; here the function `SCx2-p`(i,j,c) recognizes (non-normalized) linear combinations in $\mathbb{Z}[K^1_i \times K^2_j]$; $0^{\times}$ is the zero chain in $\mathbb{Z}[K^1_i \times K^2_j]$ for every i, j; and `Fx2-norm` is the function that in such linear combinations erases addends corresponding to degenerate pairs in the tensor product (by the way, the function `F+-norm` previously mentioned, is defined applying `Fx2-norm` in each component of the tuple).

We now present the main properties[9] establishing the Eilenberg-Zilber theorem. The following are the theorems showing that AW and EML are chain homomorphisms:

Theorem: `AW-chain-morphism`
$n \in \mathbb{N}^+ \ \wedge \ c \in C_n(K^1 \times K^2) \to AW_{n-1}(d^{\times}_n(c)) = d^{\otimes}_n(AW_n(c))$

Theorem: `EML-chain-morphism`
$n \in \mathbb{N}^+ \ \wedge \ c \in (C(K^1) \otimes C(K^2))_n \to EML_{n-1}(d^{\otimes}_n(c)) = d^{\times}_n(EML_n(c))$

And the following are the theorems establishing that (AW, EML, SH) is a reduction from $C(K^1 \times K^2)$ to $C(K^1) \otimes C(K^2)$ (properties (1) to (5) in Definition 3):

Theorem (1): `AW-EML-id`
$n \in \mathbb{N} \ \wedge \ c \in (C(K^1) \otimes C(K^2))_n \to AW_n(EML_n(c)) = c$

Theorem (2): `Cx-diff-SH-SH-Cx-diff-EML-AW-id`
$n \in \mathbb{N}^+ \ \wedge \ c \in C_n(K^1 \times K^2)$
$\to EML_n(AW_n(c)) + d^{\times}_{n+1}(SH_n(c)) + SH_{n-1}(d^{\times}_n(c)) = c$

Theorem (3): `AW-SH-null`
$n \in \mathbb{N} \ \wedge \ c \in C_n(K^1 \times K^2) \to AW_{n+1}(SH_n(c)) = 0^{\otimes}$

Theorem (4): `SH-EML-null`
$n \in \mathbb{N} \ \wedge \ c \in (C(K^1) \otimes C(K^2))_n \to SH_n(EML_n(c)) = 0^{\times}$

Theorem (5): `SH-SH-null`
$n \in \mathbb{N} \ \wedge \ c \in C_n(K^1 \times K^2) \to SH_{n+1}(SH_n(c)) = 0^{\times}$

[9]We only omit here the theorems establishing that AW, EML and SH are well-defined, which can be deduced from the well-formedness properties of the corresponding polynomial.

All these properties are obtained directly from the corresponding polynomial properties in Section 3.3, applying the well-formedness properties of the polynomials $\boldsymbol{AW}_{n,i}$, $\boldsymbol{EML}_{i,j}$ and $\boldsymbol{SH}_n$, the ring homomorphism properties of `eval-psp` and the above properties about the behaviour of `eval-psp` on cartesian and tensor degenerate polynomials. In the case of the chain morphism properties, we also need theorems relating the differentials defined in both frameworks.

4.2 Functional instantiation on a universal simplicial set

The formalization we have just presented has been done for a pair of generic simplicial sets K^1 and K^2. As defined for the formalization, the morphisms AW, EML and SH cannot be executed, since they depend on K^1 and K^2, which were introduced by the encapsulation principle. But we can instantiate the whole construction for two concrete simplicial sets, and obtain executable versions of the morphisms. In particular, we have considered the *standard simplex* Δ [16]. This simplicial set has some universal properties, since the simplicial identities are the unique constraints on it. In particular, any generic formula relating simplicial equalities will be faithfully drawn on Δ. This simplicial set is defined as follows: n-simplices in Δ are non-decreasing lists of $n+1$ natural numbers; 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. In this way, a list is a degenerate simplex in Δ_n if it contains two consecutive repeated elements.

To make the instantiation we have considered Δ as the concrete version of both generic simplicial sets K^1 and K^2, showing the correspondence between the functions of the generic formalization and those of the concrete instance. Then, the recognizer functions `K1` and `K2` are instantiated with the recognizer function of the set Δ, `Delta-K`; the simplicial operators `d1` and `d2` (respectively `n1` y `n2`) are instantiated with the face operator on Δ, `Delta-d` (respectively the degeneracy operator `Delta-n`); and the recognizer functions for degenerate simplices `K1d` and `K2d` are instantiated with the recognizer function for degenerate simplices on Δ, `Delta-Kd`. Finally, we also need to define how to instantiate the function `Kxd`, that checks if a simplex is degenerate in the Cartesian product $K^1 \times K^2$. In this case, that is $\Delta \times \Delta$, this happens when both components of a cartesian simplex have consecutive repeated elements in the same position.

Once given the functional substitution relating the generic functions and the concrete ones, the functional instantiation process builds (in an automatic way using `definstance`) concrete versions of all the remaining functions presented in the previous Section 4 and prove their properties. In this way, we have an instantiated version of the Eilenberg-Zilber theorem for the standard simplicial set Δ.

4.3 Certified programs and Kenzo running programs

Once the proof of the EZ theorem has been instantiated on the standard simplex Δ, we can evaluate the different morphisms. We concentrate on the SH operator, being the more complex. Furthermore, we can make *Kenzo* compute the same examples, and then compare both results.

The test is running over the Cartesian product $\Delta\times\Delta$, and then applied over the chain with only one monomial, with coefficient 1 and generator $((0,1,\ldots,n),(0,1,\ldots,n))$ (constructed by a function `Delta1`), belonging to $C_n(\Delta\times\Delta)$. You can find next the respective results obtained by ACL2 and by *Kenzo*, in the case $n=3$.

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

>(? shi 3 d3)
-------------------------------------------------{CMBN 4}
  <-1 * <CrPr 0 15 3-2-1 9>>
  <1 * <CrPr 1 15 3-2 11>>
  <-1 * <CrPr 1-0 7 3-2 13>>
  <-1 * <CrPr 2 15 3 15>>
  <1 * <CrPr 2-0 7 3-1 13>>
  <-1 * <CrPr 2-1 7 3 15>>
  <-1 * <CrPr 2-1-0 3 3 15>>
  <-1 * <CrPr 3-0 7 2-1 13>>
  <1 * <CrPr 3-1 7 2 15>>
  <1 * <CrPr 3-1-0 3 2 15>>
  <-1 * <CrPr 3-2-0 3 1 15>>
------------------------------------------------------------
```

Several remarks are worth mentioning. First, note the different representations used. In ACL2 a format purely list-based is employed. In *Kenzo* the internal representation of simplices in the Cartesian product is by means of a record (*struct*); the degeneracies are displayed explicitly (the string `3-2-1` stands for $\eta_3\eta_2\eta_1$), while the simplices of Δ are encoded arithmetically. For instance, the number 9 is representing the simplex $(0, 3)$, because $2^0 + 2^3 = 9$ (in general, a non-degenerate simplex $(a_0, a_1, \ldots, a_r)$ in Δ_r is represented in *Kenzo* by $\sum_{j=0}^{r} 2^{a_j}$). Thus, in the *Kenzo* term `<-1 * <CrPr 0 15 3-2-1 9>>` the first 0 is η_0 in the first factor, the number 15 ($= 2^0 + 2^1 + 2^2 + 2^3$) is representing the $(0, 1, 2, 3)$ simplex, so the first factor corresponds in ACL2 to `(0 0 1 2 3)`; therefore the first monomial in the *Kenzo* expression is denoting the term at position 7 in the ACL2 list.

The reader can check that, up to representation, both programs are computing the same element of $C_4(\Delta \times \Delta)$. Evidently we can do better than a visual inspection; we can program an automated testing. To this aim, it is necessary to apply a *domain transformation* strategy to translate from *Kenzo* format to ACL2's one. This is not difficult, and in fact the harder part was programmed (and verified) in [15], where the way of internally encoding lists of degeneracies in *Kenzo* was analyzed. It turns out that that encoding is exactly the same as for simplices in Δ, and so it is already solved.

Once *Kenzo* combinations are translated to ACL2's format, we can subtract one from another and if we get the zero combination, we ensure that ACL2 and *Kenzo* are computing exactly the same. In that way we have automatically checked that all the results around EZ (that is to say, all the computations with Alexander-Whitney, Eilenberg-MacLane and Shih morphisms) that can be computed by ACL2 coincide with those obtained from *Kenzo*.

With respect to performance, it is remarkable that the executable proof can get results up to dimension $n = 8$, in a standard laptop, before exhausting memory. On the same computer, *Kenzo* reaches dimension $n = 20$. It is necessary to point out that our ACL2 proof was not devised with efficiency in mind: it is simply the translation of the most natural mathematical ideas. In particular, *Kenzo* benefits from the compact (arithmetic) representation of degeneracies lists and of Δ simplices. This idea could be integrated in our ACL2 proof (as it was done in [15]), together with many other possible technical ACL2 improvements (compilation, guards, single-threaded objects, and so on; see [10]), getting a better ACL2 performance. We have not pursued this way, because our objective is not to compete with *Kenzo*, but building a verified counter-part that increases confidence in *Kenzo* results.

As a summary, from this experimental study we obtained clear evidence that the ACL2 proof is implementing exactly the same formulas appearing in Section 2, after the Eilenberg-Zilber theorem statement, and that the formulas are exactly the ones programmed in *Kenzo*.

5 Conclusions and future work

The Eilenberg-Zilber theorem is a central result in Simplicial Algebraic Topology, establishing a link between geometrical (Cartesian product) and algebraic (tensor product) concepts. The Eilenberg-Zilber theorem, when expressed in terms of reductions, has a companion algorithm that has been implemented in the computer algebra system *Kenzo*. In this paper, we have given a complete formal proof of the Eilenberg-Zilber theorem using the ACL2 theorem prover. Even if the formulas implemented in *Kenzo* cannot be directly translated to ACL2 (ACL2 is lacking of explicit iteration, and we are so forced to give recursive variants of the formula), experimental evidence has been provided showing that the ACL2 and the *Kenzo* implementations are behaviourally equivalent. Since the ACL2 programs are verified, trusting *Kenzo* results is now reinforced.

From a conceptual point of view, the notion of *bivariate simplicial polynomial* is the key of our approach. The simplicial polynomials machinery was also instrumental in the ACL2 proof of the Normalization Theorem [12], and it is now generalized to deal with pairs of natural transformations. The main contributions of simplicial polynomials are emulating symbolically higher-order notions (i.e. natural transformation between functors) and enhancing ACL2 with a kind of *algebraic rewriting*, that helps greatly the automation of proofs. Furthermore, executability allows unfolding recursive definitions of polynomials, and this was useful for conjecturing some lemmas which guided the proof of the main theorems.

From a technical point of view, some meta-rules have been included to deal with symbolic simplifying (so covering a potentially infinite number of simplification rules). In addition, some macros to generating instances of generic theories have been built. We hope this ACL2 technical achievements could be useful and inspiring for other developers of certified symbolic manipulation programs.

As for future work, a clear line is to apply the simplicial polynomials infrastructure to tackle other open problems in Computation Algebraic Topology, like the verification of Szczarba's twisting cochain [23] or the algorithmic solution of Adams problem on loop spaces [21]. Another research path could be to launch a project to get an efficient verified computing software for Topology; our ACL2 approach is mature enough to undertake this task. The first candidate would be the implementation of an algorithm computing the homology groups of finite simplicial sets, following ideas presented in [7].

References

[1] M. Andrés, L. Lambán, J. Rubio, J.-L. Ruiz-Reina, Formalizing Simplicial Topology in ACL2, Proceedings ACL2 Workshop 2007, University of Austin (2007) 34–39.

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

[3] J. Aransay, C. Ballarin, J. Rubio, Generating certified code from formal proofs: a case study in homological algebra, Formal Aspects of Computing 22(2) (2010) 193–213.

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

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

[6] H. Edelsbrunner and J. Harer, Computational topology: An introduction, American Mathematical Society, 2010.

[7] J. Heras, M. Dénès, G. Mata, A. Mörtberg, M. Poza, V. Siles. Towards a certified computation of homology groups for digital images, Proceedings CTIC 2012, Lecture Notes in Computer Science 7309 (2012) 49–57.

[8] J. Heras, M. Poza, J. Rubio, Verifying an algorithm computing Discrete Vector Fields for digital imaging, Calculemus 2012, Lecture Notes in Computer Science 7362 (2012) 215–229.

[9] W.A. Hunt, M. Kaufmann, R.B. Krug, J Moore and E.W. Smith. Meta Reasoning in ACL2, Procedings TPHOLs 2005, Lecture Notes in Computer Science, 3603, 163–178.

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

[11] L. Lambán, F.J. Martín-Mateos, J. Rubio, J.-L. Ruiz-Reina, Applying ACL2 to the Formalization of Algebraic Topology: Simplicial Polynomials, Proceedings ITP 2011, Lecture Notes in Computer Science 6898 (2011) 200–215.

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

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

[14] Martín-Mateos, F.J., Alonso, J.A., Hidalgo, M., Ruiz-Reina, J.L.: A Generic Instantiation Tool and a Case Study: A Generic Multiset Theory. Proceedings of the third international ACL2 workshop and its applications, (2002) 188–201.

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

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

[17] A. Prouté, Sur la Transformation d'Eilenberg-Mac Lane, Comptes Rendus Académie Sciences Paris, Série I 297 (1983) 193–194.

[18] A. Prouté, Sur la diagonale d'Alexander-Whitney, Comptes Rendus Académie Sciences Paris, Série I, 299 (9) (1984) 391–392.

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

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

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

[22] J. Rubio, F. Sergeraert, Y. Siret, EAT: Symbolic Software for Effective Homology Computation, Institut Fourier, 1997.
`http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/#Eat`

[23] R. H. Szczarba, The homology of twisted cartesian products, Transactions of the American Mathematical Society, 100 (1961) 197–216.