Effective Homology of Bicomplexes, formalized in Coq

César Domínguez*, Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja. Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain).

Abstract

In this paper we present a complete formalization in the Coq theorem prover of an important algorithm in computational algebra, namely the calculation of the effective homology of a bicomplex. As a necessary tool we encode a hierarchy of algebraic structures in constructive type theory, including graded and infinite data structures. The experience shows how some limitations of the Coq proof assistant to deal with this kind of algebraic data can be overcome by applying a separation of concerns principle; more concretely, we propose to distinguish in the representation of an algebraic structure (as a group or a module) a behavioural part, containing operation signatures and axioms, and a structural part determining if the algebraic data is free, of finite type and so on.

Keywords: Theorem proving, formal methods, computer algebra, program verification.

1. Introduction

As signaled by Broy in [6] formal methods are nowadays powerful enough to deal with real life systems. To give only two examples, the Verisoft project (see [5], for instance) and the project to build a C compiler by means of the Coq proof assistant (see [19]) are demonstrating Broy's claim. At the same time (and we think this is not something accidental) the mechanized reasoning tools are now ready to tackle non trivial mathematical theorems, as in the case of Hales' proof of Kepler conjecture (through the Flyspeck project,

*Corresponding author.

Email addresses: cesar.dominguez@unirioja.es (César Domínguez), julio.rubio@unirioja.es (Julio Rubio)

see [21]) or Gonthier's proof in Coq of the Four Colour Theorem (see [16]). And, somewhere in the middle between theorem proving applied to software engineering and to the formalization of mathematics, we find the use of these tools to verify mathematical software, as in the project Calculemus [7] where the emphasis is put in the integration of Proof Assistant and Computer Algebra systems.

This is the context where our research is located. More concretely, in the area of Homological Algebra and Algebraic Topology computing systems. At the end of the eighties, Sergeraert introduced a systematic approach to produce computational results in these fields of mathematics, called *effective homology* [24]. Nowadays, there exists an increasing interest on this kind of computational Algebraic Topology, mainly due to its applications (to digital image analysis, data mining and other disciplines); see for instance in [12] an introduction to this research. A characteristic feature of Algebraic Topology is that, in some situations, to obtain results on *finite* spaces, it is necessary to pass as intermediary steps through *infinite dimensional* spaces. Sergeraert's effective homology explicitly deals with these infinite spaces. In particular, it was clear from the beginning the importance of determining the effective homology of bicomplexes (as a way to find out algorithmic substitutes to spectral sequences). In 1988 such an algorithm was presented in [22], based on the notion of *cone* of a morphism. Later on, the role of a result called *Basic Perturbation Lemma* (or *BPL*, in short) was discovered. Almost all the algorithms used previously in the effective homology setting could be replaced by particular instances of the BPL. As a consequence, the development of programs was dramatically accelerated, given rise to the computer algebra system called *Kenzo* [14]. This system is the first one in dealing with complex infinite dimensional constructions (as loops spaces or classifying spaces), being capable of computing homology groups previously unknown, and whose validity cannot therefore be confirmed nor refuted by any other means.

And then formal methods start playing a role, analyzing and verifying parts of Kenzo. In a first step, algebraic specification techniques were used to study the data structures appearing in Kenzo (see [18, 10, 11], for instance). Once the structural aspects were formally understood, theorem provers were employed to verify the *algorithms* implemented in Kenzo. The first important milestone in this area was the mechanized proof in the Isabelle/HOL proof assistant of the Basic Perturbation Lemma, published in [1]. This formal proof was carried out in the Higher Order Logic (HOL) built on top of

Isabelle, and therefore extracting programs from it was not a simple task. The findings on this topic were reported in [2].

Another alternative to get certified programs in this setting is to move to constructive mathematics, and more concretely to constructive type theory by encoding our proofs in the Coq assistant [4]. Some tentative works and comparisons between Isabelle and Coq [3] made clear that it is possible. Nevertheless, we consider unnecessary to repeat the complete BPL proof in Coq. The reason is that, since it was already implemented in Isabelle by using all power of classical logic (working explicitly, for instance, with general sets and subsets of them), the translation to Coq would be unnatural. A different approach is being carried out by Coquand and Spiwack [9] who are using Coq to model a part of Category Theory, and then trying to obtain a BPL proof in this larger context.

Our aim in this paper is different. We have gone back to the effective homology origins, and we have implemented in Coq the algorithms based in cones previously evoked (published in 1988 in [22]). We have found that this high level algorithm has an inductive structure that can be translated very naturally to Coq, allowing us to perform a complete Coq implementation of a correct algorithm computing the effective homology of a bicomplex. This result covers most of the applications of the BPL in Kenzo, and thus its interest is not negligible from the verification point of view. Another benefit we obtain from our formalization is the treatment of graded and infinite dimensional data structures, which were missing in the previous Isabelle mechanization of the BPL. Even if our endeavour is very concrete, the infrastructure we developed is quite general, dealing with a complex hierarchy of algebraic data structures. As an illustration of this generic aspect, let us observe that our work meets several situations shared with the project by Gonthier and collaborators [17] to devise in Coq a modular formalization of Finite Group Theory.

The organization of the rest of the paper is as follows. In the next section, we introduce some mathematical preliminaries, which are encoded in Coq in Section 3. The effective homology of a cone is implemented in Coq in Section 4, while Section 5 is devoted to comment on a hierarchy of algebraic data structures. The previous tools are then used in Section 6 to establish, in a simple inductive manner, our main result: the theorem about the effective homology of a bicomplex. The paper ends with a section of Conclusions and Further Work, and the bibliography. The Coq file sources are available at: `http://www.unirioja.es/cu/cedomin/EHBFC.zip`.

2. Preliminaries

In this section we define the algebraic structures needed in our formalization. They include, in particular, chain complexes, reductions and effective homologies of chain complexes.

We assume as known the notions of *ring*, *module* over a ring and *module morphism* (see [13] for instance). A ring R commutative and with unity is fixed along all the paper, and modules are supposed to be *left* R-modules.

Given a set B, we define a new set denoted by $R[B]$ whose elements are linear combinations with elements of B as generators. That is to say, the elements of $R[B]$ are lists of terms, where a term is a pair composed of a *coefficient*, a non-null element of R, and a generator, an element of B. Emulating the way of working in linear algebra, $R[B]$ can be naturally endowed with an addition and an external product by elements of R, allowing us to consider a structure of R-module on $R[B]$. This module is called the *free R-module generated over B*. Since we are planning to work in a constructive logic setting, it is convenient to define a *free* module as one module M where an explicit isomorphism is known among M and $R[B]$ (the set of generators B must be also explicitly given).

Let us note that the set B is not necessarily finite, because, as mentioned in the introduction, Algebraic Topology and so the Kenzo system need to deal with infinite dimensional structures. In order to deal with finite sets in a constructive type theory, even more care is needed. For instance, several alternatives for defining finite sets in a constructive logic are included in [8]. Finite algebraic structures have also been implemented in Coq in [17] as the first milestone of a long-term effort to formalize the Feit-Thompson theorem.

Our definition runs as follows. Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) set $\{0, 1, \ldots, k-1\}$. Then, we consider a set B as *finite* if it is endowed with a natural number $k \in \mathbb{N}$ and an explicit bijection $\psi : B \to FS(k)$ with an explicit inverse $\psi^{-1} : FS(k) \to B$. A free R-module is *of finite type* if it is explicitly isomorphic to $R[B]$ with B a finite set in the previous sense.

We are ready to introduce the first *graded* concept, needed in Homological Algebra and Algebraic Topology.

Definition 1. *A (positive)* graded module *M is a family of R-modules $(M_n)_{n\in\mathbb{N}}$. A graded module is* free *(or* free of finite type*) if every M_n is free (free of finite type, respectively) for all $n \in \mathbb{N}$.*

Usually, graded modules are indexed by the integers $\mathbb{Z}$, but it is more convenient for us to work with *positive* graded modules (indexed by $\mathbb{N}$), as in Coq it allows us to work with the comfortable data type `nat` for representing indexes. Furthermore, to deal with positively graded structures is enough for our applications in this paper.

Definition 2. *Given a graded module M a* differential operator *d on M is a family of module morphisms $(d_n \colon M_{n+1} \to M_n)_{n\in\mathbb{N}}$ such that $d_n \circ d_{n+1} = 0$ for all $n \in \mathbb{N}$.*

Again, the convention of denoting the differential d_n as having source M_{n+1} will allow us to work in Coq within `nat`.

Definition 3. *A* chain complex *is a pair $CC = (M, d)$ where M is a graded module and d a differential operator on M. A chain complex is called* free *(or* free of finite type*) when its underlying graded module is free (free of finite type, respectively).*

Chain complexes have a corresponding notion of morphism:

Definition 4. *A* chain complex morphism *(or, simply, a* chain morphism*) $f \colon CC \to CC'$ between two chain complexes $CC = (M, d)$ and $CC' = (M', d')$ is a family of module morphisms $(f_n \colon M_n \to M'_n)_{n\in\mathbb{N}}$ such that $f_n \circ d_n = d'_n \circ f_{n+1}$ for all $n \in \mathbb{N}$.*

Now, the central definition in effective homology theory: *reduction*. A reduction establishes a link between a "big" chain complex, called *top complex*, and a smaller one, called *bottom complex*, in such a way that if all the homological problems are solved in the bottom complex, then it is the same in the top one. A solution for a homological problem in a chain complex (M, d) is a series of procedures (algorithms, in the computational setting) asking questions as: is an element $x \in M_n$ a boundary (*i.e.* $x \in Im(d_n)$)? If it is the case, can you produce an element $y \in M_{n+1}$ such that $x = d_n(y)$? Answering these questions is one of the most important problems in Homological Algebra and, as a consequence, in Algebraic Topology. See [23] for details.

Definition 5. *A* reduction *is a 5-tuple (TCC, BCC, f, g, h) where $TCC = (M, d)$ and $BCC = (M', d')$ are chain complexes (named* top *and* bottom

chain complexes, respectively), $f\colon TCC \to BCC$ *and* $g\colon BCC \to TCC$ *are chain morphisms,* $h = (h_n\colon M_n \to M_{n+1})_{n\in\mathbb{N}}$ *is a family of module morphisms (called* homotopy operator*), which satisfy the following properties for all* $n \in \mathbb{N}$*:*

1. $f_n \circ g_n = id_{M'_n}$
2. $d_{n+1} \circ h_{n+1} + h_n \circ d_n + g_{n+1} \circ f_{n+1} = id_{M_{n+1}}$ *and* $d_0 \circ h_0 + g_0 \circ f_0 = id_{M_0}$
3. $f_{n+1} \circ h_n = 0$
4. $h_n \circ g_n = 0$
5. $h_{n+1} \circ h_n = 0$

And now, the relevant case. In a free chain complex *of finite type* the homological problems can be solved algorithmically in a simple way (at least in cases where the ring R allows one to diagonalize matrices over R; this includes the case $R = \mathbb{Z}$, the most important one in Algebraic Topology; see [23]). Thus, if from a chain complex (possibly of infinite type) we can get a reduction to a chain complex of finite type, the homological problem is solved for the initial complex. This is the strategy followed in the Kenzo system. And it is the very notion of chain complex with *effective homology.*

Definition 6. *A chain complex CC is with (strong)* effective homology *if it is free and it is endowed with a reduction where CC itself is the top chain complex and the bottom chain complex is free* of finite type.

The adjective "strong" appears since it is a particular case of the general notion of effective homology (see [23]). Nevertheless, solving the problems for this kind of effective homology, the general case follows easily.

3. Coq infrastructure

Although the standard library of Coq [20] does not include a representation for basic algebraic structures, different implementations of them can be found in the literature. For instance, there are two representations, CoRN (Constructive Coq Repository at Nijmegen; see also [15]) and Algebra (by L. Pottier), published in the users' contributions in [20]. Even if both formalizations have different proposals and natures (the first one is an axiomatic hierarchy of the most common algebraic structures based on setoids with an apartness relation; the second one has a categorical flavor), both represent

these algebraic structures using records. We have built our own structures based on the ones included in CoRN (but simplifying them: basically eliminating the apartness relation included in setoids which is not used by us, since we are working in a *discrete* mathematics setting). A *setoid* is a Coq type together with an equivalence relation defined on it (the *equality* of the setoid). The CoRN repository includes rings, modules (over a ring) and module morphisms implemented as records called `Ring`, `Module` and `ModHom`, respectively. Besides, further constructions as for instance the addition or the composition of module morphisms are defined, and are represented using the infix notation: `[+h]` or `[oh]`, respectively. The repository also contains useful lemmas on these structures (see [20] for a detailed description).

We concentrate ourselves in the sequel on *free* modules, since it is the unique kind of modules dealt with in the Kenzo system [14].

The formalization of free modules follow the ideas given by L. Pottier in the Coq contributions web page [20]. There, a definition can be found of a module defined by freely generation from a basis (which is given by a setoid) using the module operations. If we call B the basis setoid, this is representing the mathematical structure $R[B]$ introduced in the previous section. Then, as explained there, a free module is a module with an explicit isomorphism to such a freely generated module. For free modules of finite type, we change the view: a free module of finite type is simply a free module, but we impose that the generator set is *equal* to a *standard* finite setoid $FS(k) = \{0, 1, \ldots, k-1\}$, as mentioned before. With this encoding we get a separation of concerns: structurally, free chain complexes and free chain complexes of finite type are indistinguishable (as in mathematics!), but a new property is imposed to finite type modules. See Section 5 for a more detailed discussion on the consequences of this design decision.

Given a ring `R: Ring`, a free graded module can be formalized in Coq with the following *dependent* type: `nat -> FreeModule R`, which accurately represents a family of free modules indexed by the natural numbers. Then, a (positive) *free* chain complex can be formalized in Coq using the following record structure:

```
Record ChainComplex: Type:=
 {GrdMod:> nat -> FreeModule R;
  Diff: forall n:nat, ModHom (R:=R) (GrdMod (S n)) (GrdMod n);
  NilpotencyDiff: forall n:nat, (Nilpotency (Diff n)(Diff (S n))}.
```

where the nilpotency property is defined by `Nilpotency(g:ModHom B C)(f: ModHom A B):= forall a:A, ((g[oh]f) a)[=]Zero`.

Some comments on the above record structure are required. First, we define only *free* chain complexes because in our applications it is enough to work with this kind of complexes (note the occurrence of the `FreeModule` identifier in the `GrdMod` field). Second, the algebraic laws in definitions must be compatible with the underlying *setoid* structure, *i.e.*, with its equality denoted by `[=]` in the previous formulae. Third, the `GrdMod` component is declared (by the annotation `:>`) to be a *coercion function*. This means that the type checker will insert this function over a chain complex when a graded module is required. These techniques are extensively used in the implementation of algebraic structures [20].

In a similar way, given two chain complexes `CC1 CC2:ChainComplex R`, a chain complex morphism `ChainComplexHom` is represented as a record with a family of module morphisms `GrdModHom:> forall n:nat, ModHom (CC1 n)(CC2 n)` which commutes with the chain complex differentials. A homotopy operator is defined as a family of module morphisms `HomotopyOperator := forall n:nat, ModHom (CC1 n)(CC1 (S n))`.

Then, the *reduction* notion is also formalized as a record `Reduction` with two chain complexes `topCC, bottomCC: ChainComplex R`, and three morphisms `f_t_b:ChainComplexHom topCC bottomCC`, `g_b_t:ChainComplexHom bottomCC topCC`, `h_t_t:HomotopyOperator topCC`. Besides, five fields representing the five reduction properties are included. For instance, the field which corresponds to the second property is: `rp2:homotopy_operator f_t_b g_b_t h_t_t` with:

```
Definition homotopy_operator:= forall (n:nat)(a:CC1 (S n)),
 ((Diff CC1 (S n) [oh] h(S n))[+h](h n [oh] Diff CC1 n)[+h]
      (g(S n)[oh] f(S n)))a [=] a
  /\ forall (a:CC1 0), ((Diff CC1 0 [oh] h 0) [+h]
      (g 0[oh] f 0))a [=] a.
```

The concept of *Effective Homology* is then obtained as a specialization of the *reduction* structure: simply declaring that the `bottomCC` is of finite type, without changing the structure of the record.

4. Effective homology of a cone

Before defining the notion of mapping cone, the concept of *suspension* is required.

Definition 7. *Given a chain complex* $M = ((M_n)_{n\in\mathbb{N}}, (d_n)_{n\in\mathbb{N}})$*, the* suspension *of* M *is the chain complex* $S(M) = ((S(M)_n)_{n\in\mathbb{N}}, (S(d)_n)_{n\in\mathbb{N}})$ *such that,* $S(M)_{n+1} = M_n$ *and* $S(d)_{n+1} = d_n$ *for all* $n \in \mathbb{N}$*. The chain complex* $S(M)$ *is completed with null components in dimension* 0*. The suspension definition can be naturally extended to chain morphisms and homotopy operators.*

In Coq, the suspension of a free *graded module* `GM: GrdFreeModule R` is formalized as follows.

```
Definition Susp_GrdFreeMod: GrdFreeModule R:= fun n:nat =>
 match n with
 | 0 => NullFreeModule R
 | S n => GM n
 end.
```

It is naturally extended in Coq to suspensions of chain complexes (`Susp_CC`), chain morphisms (`Susp_CC_Hom`), and homotopy operators (`Susp_HO`).

Definition 8. *Given a pair of chain complexes* $CC = ((M_n)_{n\in\mathbb{N}}, (d_n)_{n\in\mathbb{N}})$ *and* $CC' = ((M'_n)_{n\in\mathbb{N}}, (d'_n)_{n\in\mathbb{N}})$ *and a chain morphism* $\alpha\colon CC \to CC'$*, the* cone *of* α*, denoted by* $Cone(\alpha)$*, is a chain complex* $CC'' = ((M''_n)_{n\in\mathbb{N}}, (d''_n)_{n\in\mathbb{N}})$ *such that, for each* $n \in \mathbb{N}$*,* $M''_n = S(M)_n \oplus M'_n$ *and* $d''_n(x, x') = (-S(d)_n(x), d'_n(x') + \alpha_n(x))$ *for any* $x \in S(M)_{n+1}$ *and* $x' \in M'_{n+1}$*.*

The relevant part of the corresponding Coq definition is:

```
Definition ConeDiffGrdMod:= fun(n:nat)(ab:(ConeGrdMod (S n))) =>
  ([--](Diff (Susp_CC CC1) n (fst ab)),
  (Diff CC0 n)(snd ab) [+] alpha n (fst ab)).
```

It is not difficult to prove that these Coq functions define a module morphism which verifies the nilpotency condition. This last property allows us to build the cone chain complex associated to a chain morphism: `Cone(alpha)`.

Let us observe that if a graded module is free, its suspension is also free. It is the same for the *direct sum* of two free graded modules, and the same applies if we replaced *free* by *free of finite type*. As a consequence, if a morphism α is defined between free chain complexes (or free chain complexes of finite type), then $Cone(\alpha)$ is a *free* chain complex (free chain complex *of finite type*, respectively). These remarks, and the form of the statement, give an elementary proof of the following result.

Theorem 1. *Given two reductions* $r = (TCC, BCC, f, g, h)$ *and* $r' = (TCC', BCC', f', g', h')$ *and a chain morphism* $\alpha\colon TCC \to TCC'$ *between their top chain complexes, it is possible to define a reduction* $r'' = (Cone(\alpha), BCC'', f'', g'', h'')$ *with* $Cone(\alpha)$ *as top chain complex and:*

- $BCC'' = Cone(\alpha')$ *with* $\alpha'\colon BCC \to BCC'$ *defined by* $\alpha' = f' \circ \alpha \circ g$
- $f'' = (f, f' \circ \alpha \circ h + f')$, $g'' = (g, -h' \circ \alpha \circ g + g')$, $h'' = (-h, h' \circ \alpha \circ h + h')$

Besides, if TCC *and* TCC' *are objects with effective homology through the reductions* r *and* r'*, then* $Cone(\alpha)$ *is an object with effective homology through* r''*.*

Observe that the statement reflects the way of working (and thinking) in mathematics: all the structural part is proved by means of the *reduction* notion. Then, as a corollary, the effective homology version is obtained, by simply remarking that the suspension and cone on finite type modules, continue to be of finite type. Our Coq infrastructure allows us to mimic this way of working, without repeating (an essentially equivalent) proof for the effective homology case.

The proof is translated to Coq, by using the previous defined types. For instance, the first chain morphism of the reduction is:

```
Definition f_coneGrdMod:
forall n:nat, (Cone alpha) n -> (Cone alpha') n:=
 fun(n:nat)(ab:(Cone alpha) n)) =>
   (Susp_CC_Hom (f_t_b r1) n (fst ab),
      ((f_t_b r2) n [oh] alpha n [oh]
       Susp_HO (h_t_t r1) n)(fst ab) [+] f_t_b r2 n (snd ab)).
```

5. A hierarchy of algebraic data structures

Even if the previous proof can be carried out without any special problem in Coq, the infrastructure needed to prove it deserves some explanation. It depends heavily on a hierarchy of algebraic data structures. The devising of such a hierarchy, and the consequences in terms of further proof effort, are discussed in this section.

Our first attempt to develop the proofs was based on the idea of keeping a double hierarchy of Coq structures: one dealing with free modules, and the other one with free modules of finite type. It is true in mathematics, but

when translating the idea to Coq we found some shortcomings. The reason is that a free module has a field containing its generator set (a Coq setoid). Then a free module of finite type contains a finite setoid. A finite setoid *is* a setoid in Coq (by means of an implicit coercion), but it is not possible to keep the same relation between a free module and a free module of finite type, since an implicit coercion will include *two* setoid structures on a free module of finite type. The situation is similar to multiple inheritance, even if from a conceptual (mathematical) point of view it is different. Therefore, we were obligated to maintain separated the two basic structures of free module and free module of finite type. The definition and proofs (for suspensions, cones and the like) must then be repeated (even if only the name of structures changed). Even worse, this approach led us to a multiplication of morphisms: in order to define the concept of effective homology, we had to consider morphisms between free modules, from a free module to a free module of finite type, etc. Four kinds of morphisms were needed, even if, structurally, they are exactly the same (and, therefore, it is the same for their properties and proofs). In addition, if we want to prove first the results for reductions, then we must replay the same proofs for the cases of effective homologies. It is possible, and it is not difficult, but the deficiencies from the elegance and the proof engineering points of view are evident. Nevertheless we got a first complete version of all the results of the paper (including the effective homology of bicomplexes of next section) with this cumbersome strategy.

Once this first version was finished, we tried to simplify the structure of our proof, without redoing it. That is to say, we changed some definitions in records, but keeping essentially the tactic steps in proofs. The first modification was to link free modules of finite type with free modules by means of an *explicit* coercion. This allows us to avoid the shortcomings of implicit coercions previously evoked, and also permits decreasing the duplication of definitions. In particular, we consider only a kind of morphism, without distinguishing the types of the modules in the source and the target of each morphism. And in a third, and final, simplification we get rid of the repetitions in statements and proofs for reductions and, then for effective homologies, by the direct mechanism of ignoring reductions. This makes sense since all our main results are related to effective homology, acting the concept of reduction as an intermediary notion, that can be skipped. However the price to be paid is expensive: our results were less general (so, making difficult their reusing in other close formalizations) and the way of working as in "paper and pencil" mathematics was lost.

Even with these severe simplifications, the new proof was not either fully satisfactory. It was necessary to duplicate some proofs related to suspensions (and, from them, those related to cones, bicomplexes, etc.). The deep reason is that, when dealing with modules of finite type, the suspension of the coercion (from a module of finite type to a module) is not Coq convertible to the coercion of a suspension. And therefore an explicit management of the two cases was needed (even if proofs are *exactly* equal).

Another alternative considered was to redefine the hierarchy as a whole in a *parameterized* style, like in the new architecture of the CoRN repository [25]. Nevertheless, the same difficulty related to coercions and suspensions was found in our first experiences with this approach and, in addition, the complete changing of organization did not allow us to reuse the previous proofs, needing to rewrite them from scratch. This via should be explored further to conclude if these difficulties can be overcome.

Finally, the solution found was the following. We keep exactly the same structure (the same record) for free modules and free modules of finite type, but for the second ones we specialize their behaviour by imposing that the generator setoid B *is equal to* a finite setoid $FS(k) = \{0, 1, \ldots, k-1\}$. Let us stress that here "equal to" refers to *Leibniz* equality, since in our development we are applying the property of replacement only for propositions. With this small changing of view (before, the record for a free module of finite type had a slot of type finite setoid, different from the setoid slot of a free module), the structure of the proofs becomes much simpler. In particular, a separation of concerns principle can be used in writing proofs: first, we build a proof for modules, and then we specialized it for modules of finite type, *adding simply a new fragment, without touching at the previous proof.* For instance, to prove that the cone of a morphism between chain complexes of finite type is a chain complex of finite type, we first prove that it is a chain complex, and then, by simply observing that the sum of modules of finite type is still of finite type, we finish the proof.

In Figure 1 we show the part of the hierarchy dealing with free chain complexes. It can be considered a simple extension of the CoRN hierachy to *graded* algebraic structures. A continue arrow describes an implicit coercion whereas a dashed arrow means a *use* relationship in the sense that a type appears in the definition of another type, but excluding coercions. Figure 2 displays the part of the hierarchy related to finitely generated structures. As it can be remarked, this last part is simpler and illustrates a general way to add new features to an already-built hierarchy in a modular way.

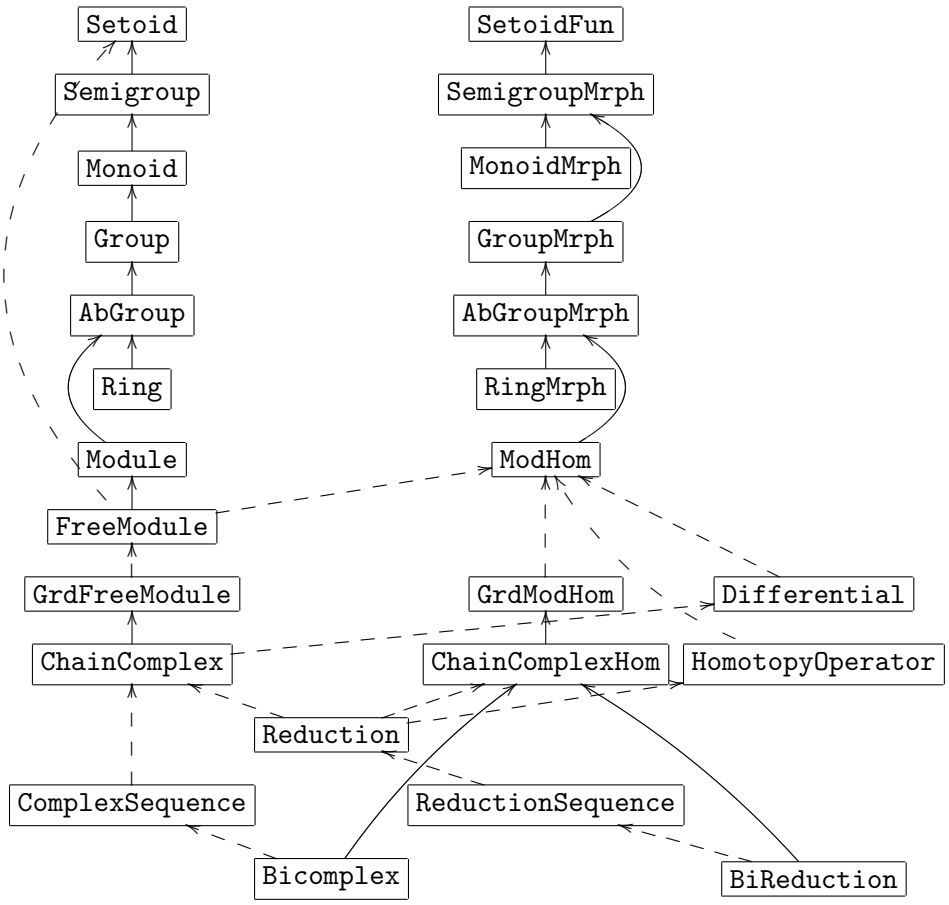

Figure 1: Free algebraic structure hierarchy

Our way of organising finally the proof is inspired by previous work in the Isabelle/HOL proof assistant (see [1] and [2]). In that proving environment types (structure) and properties (behaviour) are uncoupled, easing the translation of the working mathematician style to the computer. Of course, the absence of dependent types (and, in general, the poorer expressiveness of the type system) produces proofs where more constraints must be encoded in the logical part, in order to avoid inconsistent expressions that the Coq type system does not allow. With our treatment of free modules of finite type, we think we obtain the best of both worlds: the proofs can follow accurately the textbooks guidelines, and the full power of types is applied in

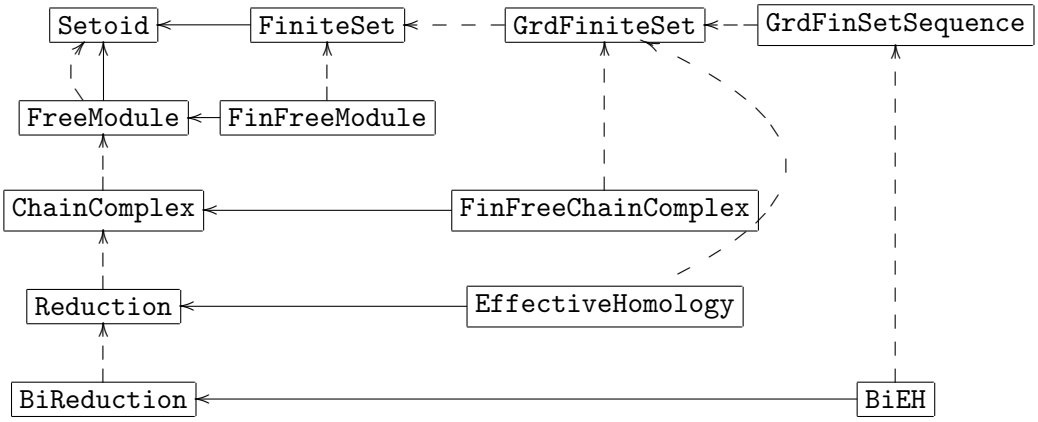

Figure 2: Free of finite type algebraic structure hierarchy

the development.

6. Effective homology of a bicomplex

In this last, and main, section we are going to use the previous infrastructure and the formalization of mapping cones to build a Coq proof of the effective homology of a bicomplex. Let us first introduce smoothly this central notion in homological algebra, by giving some auxiliary definition. We are going to work with *sequences* (that is to say, families indexed by $\mathbb{N}$) of the previously introduced objects.

Thus, a *sequence of graded modules* is a map from $\mathbb{N}$ to the class of graded modules. Let us denote by $B = \{B_{p,*}\}_{p\in\mathbb{N}}$ such a sequence. If we denote by q the index in each graded module, the information contained in the previous definition is exactly the same as in the following one.

Definition 9. *A* bigraded module B *is a system* $B = \{B_{p,q}\}_{p,q\in\mathbb{N}}$ *where every* $B_{p,q}$*,* $p, q \in \mathbb{N}$*, is a module.*

From now on, both concepts are identified. A bigraded module B is called *free* (or *free of finite type*) when all the modules $B_{p,q}$ are free (free of finite type, respectively).

Each sequence of objects comes endowed with a corresponding notion of *totalization*.

Definition 10. *Given a bigraded module* $B = \{B_{p,q}\}_{p,q\in\mathbb{N}}$*, the* totalization *of* B *is the graded module* $T(B) = \{T(B)_n\}_{n\in\mathbb{N}}$ *where* $T(B)_n = \oplus_{p+q=n} B_{p,q}$*.*

It is clear that the totalization of a *free* bigraded module is a *free* graded module. More care is needed to prove that the totalization of a bigraded module free *of finite type* is also free *of finite type*. This is true because, given a natural number $n \in \mathbb{N}$, only finitely many pairs $(p, q) \in \mathbb{N} \times \mathbb{N}$ satisfy $p + q = n$. It is not longer true, for instance, if the sequences of graded modules were indexed by the integer numbers $\mathbb{Z}$. It is the reason why in the literature our bigraded modules are called more specifically *first quadrant* bigraded modules.

The next notions to be considered are those of sequence of chain complexes and of chain morphisms. If an additional condition is required to a sequence of chain morphisms, all the information can be totalized in a unique chain complex. This is collected in the following definition of a *bicomplex*.

Definition 11. *A (first quadrant)* bicomplex B *is a pair* $((B_{p,*})_{p\in\mathbb{N}}, (b_p)_{p\in\mathbb{N}})$ *with* $(B_{p,*})_{p\in\mathbb{N}}$ *a chain complex sequence and* $(b_p \colon B_{p+1,*} \to B_{p,*})_{p\in\mathbb{N}}$ *a sequence of chain morphisms, such that* $b_p \circ b_{p+1} = 0$*.*

The corresponding type in Coq looks as follows.

```
Record Bicomplex: Type:=
{FCC: nat -> ChainComplex R;
 FCCh:> forall (n:nat), ChainComplexHom (FCC(S n))(FCC n);
 NilpFCCh: forall (n m:nat), (Nilpotency(FCCh n m)(FCCh(S n) m))}.
```

The "nilpotency" condition $b_p \circ b_{p+1} = 0$ allows us to prove that the totalization of a bicomplex is really a chain complex.

Definition 12. *Given a bicomplex* $B = ((B_{p,*}, d_{p,*})_{p\in\mathbb{N}}, (b_{p,*})_{p\in\mathbb{N}})$*, the* totalization *of* B *is the chain complex* $T(B) = ((T(B)_n)_{n\in\mathbb{N}}, (d_n)_{n\in\mathbb{N}})$ *where* $T(B)_n = \oplus_{p+q=n} B_{p,q}$ *and* $d_n = \oplus_{p+q=n}((-1)^p d_{p,q} \oplus b_{p,q})$*.*

Let us note that the underlying graded module $T(B)$ is exactly the totalization of the underlying bigraded module $\{B_{p,q}\}_{p,q\in\mathbb{N}}$.

Imagine now that we have a *degenerated* bicomplex B where $B_p = 0$, $\forall p > 1$. In other words, B is only a chain morphism $b_0 \colon B_1 \to B_0$. Then, it is easy to check that the totalization $T(B)$ is exactly $Cone(b_0)$. Thus, mapping cones are totalizations of a particular kind of bicomplexes. In the sequel,

we will use that the totalization of each bicomplex can be understood as an iteration of mapping cones. Then, by iterating the algorithm to compute the effective homology of a cone, we will get the corresponding result on the effective homology of a bicomplex.

The first step in order to formalize the totalization of a bicomplex is to consider in Coq the cone of the first morphism in a bicomplex `F:Bicomplex` which is simply obtained by `Cone1:=Cone (F 0)`. Then, we can easily define a new sequence of chain complexes:

```
Definition new_ComplexSequence:=
 fun n : nat => match n with
   | 0 => Cone1
   | S n => Minus_ChainComplex(Susp_CC(FCC F(S(S(n)))))
 end.
```

and similarly a new bicomplex through a family of chain morphisms.

This construction can be endowed with an iterator, in such a way that at step n in the iteration, the module in degree n of the first chain complex of the family coincides with the module in degree n of the totalization of the initial bicomplex. The same applies to the differential d_n in the totalization.

In that way, (totalizations of) bicomplexes can be understood as iterated cones. In the statement of the theorem about the effective homology of a cone, it is explained that the output bottom complex is again the cone of a certain morphism. Then it is appealing to think that the bottom complex in the effective homology of a bicomplex could be also (the totalization of) a bicomplex. Nevertheless it is not true: even if a bicomplex can be interpreted as an iteration of cones, in general an iteration of cones does not define a bicomplex, but a more complicated structure called *multicomplex* (the reason is that, in a bicomplex, the target of a morphism is always in the first "column" of a cone; on the contrary, a general morphism arriving to a cone can have part of its image in each "column" of the cone). We do not need to introduce here the general notion of *multicomplex*, because it will be automatically produced by our mechanical Coq proof of the main result, whose statement is as follows.

Definition 13. *A bicomplex B is an object with effective homology if its totalization $T(B)$ is a chain complex with effective homology.*

Theorem 2. *Let $B = ((B_p)_{p\in\mathbb{N}}, (b_p)_{p\in\mathbb{N}})$ be a bicomplex such that each chain complex B_p is with effective homology, for all $p \in \mathbb{N}$. Then the bicomplex B is an object with effective homology.*

As in the case of the Theorem 1 for the homology of the cone, we will prove first the structural part of the theorem working with reductions. Then, the effective homology version is obtained.

Let us call `BiReduction` the Coq data structure consisting in a sequence of reductions together with a sequence of chain morphisms between the *top* chain complexes satisfying the nilpotency condition (in other words, the top chain complexes together with the chain morphisms form a bicomplex). Then, we repeat with it the iterative process previously done for a single bicomplex, using now as starting step the Coq theorem on the reduction of the cone. If we denote by `new_BiReduction` the corresponding iterative step, then an iterator can be defined in Coq as follows:

```
Fixpoint iterated_BiReduction
         (n:nat)(F:BiReduction){struct n}: BiReduction:=
 match n with
  |0 => F
  |S n => New_BiReduction (iterated_BiReduction n F)
 end.
```

We have now all the necessary instruments to prove in Coq our main theorem. The harder part is to deal with the following dependent type:

```
Definition Diff_bottom_totalization: forall n: nat,
ModHom (bottomCC (FR (iterated_BiReduction (S n) F) 0) (S n))
(bottomCC (FR (iterated_BiReduction n F) 0) n):=
 fun n:nat => sndConeDiff(alpha'((iterated_BiReduction F) 0)) n.
```

and then to prove that it defines really a chain complex (as mentioned before, it is the differential of the totalization of a *multicomplex*, but here it is produced automatically as the iteration of the reduction of cones).

Then it is necessary to prove that this chain complex appears as the bottom complex of a reduction where the top complex is the (totalization of the) input bicomplex (it is again a corollary of the corresponding result for mapping cones).

To finish the proof it is necessary to ensure that, if we take from the premise as input a `BiEH` structure, *i.e.* a `BiReduction` whose sequence of reductions are indeed effective homologies, the previous bottom chain complex is of finite type. But here it is enough to observe that the underlying graded module is the totalization of a (first quadrant) bigraded module of finite type (it is enough to observe it, but ... it is necessary to convince Coq of it, too!).

7. Conclusions and further work

In this paper, a complete formalization of an important algorithm in computational algebra (namely, the effective homology of a bicomplex) has been presented. The proof assistant used has been Coq, and thus our work can be understood as the application of constructive type theory to a piece of sophisticated mathematics. The verified algorithm is related to a Computer Algebra system for Algebraic Topology called Kenzo [14]. Therefore, our research is placed between the efforts to formalize mathematics and the application of formal methods in software engineering.

Interestingly enough, we did not re-elaborated on a previous work of formalization in this area carried out in Isabelle/HOL [1, 2]. We have retaken an older algorithm [22] which has demonstrated to be better adapted to the type theoretic Coq style. As a comparison between the two proof assistants, we can conclude that the dependent types in Coq allow the modeler a most accurate representation of the mathematical structures (on the contrary, the rigidities of constructive type theory make difficult to translate to Coq the more set-theoretic argumentation implemented in [1] by means of Isabelle/HOL).

Perhaps the contribution that could be more useful for other researchers is the reflection on how building complex hierarchies of mathematical structures (needed in most of the formalization projects on non-trivial mathematics; see [15, 17], for instance), and how it relates to the further proving effort. Our conclusion is that Coq needs more automation in this area, beyond of inheritance (single coercion or multiple inheritance), to emulate the way working mathematicians deal with their objects of study.

This task can be considered as an open problem, and it is a first line of future work. Another topic is related to the executability of our proofs. Working in a constructive type theoretic setting, it is always possible to reduce terms inside Coq itself, but it is needed first to get concrete instances of any concept (to deal with meaningful examples), and it is not always easy to obtain, specially in our context where data structures of infinite type occur. Additionally, even if this obstacle is overcome, the question of performance would still be open. Being Kenzo a Common Lisp system, we are planning to extract Common Lisp code from our Coq proofs, in order to compare it with the original programs, focus of our study.

Acknowledgements

Partially supported by Ministerio de Educación y Ciencia, project MTM 2009-13842-C02-01, and by the FORMATH project, nr. 243847, of the FET program within the 7th Framework program of the European Commission.

References

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

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

[3] Aransay, J., Domínguez, C.: Modelling Differential Structures in Proof Assistants: The Graded Case. *Eurocast 2009*, Lecture Notes in Computer Science 5717 (2009) 203-210.

[4] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Springer, 2004.

[5] Botaschanjan, J., Broy, M., Gruler, A., Harhurin, A., Knapp, S., Kof, L., Paul, W.J., Spichkova, M.: On the correctness of upper layers of automotive systems. Formal Aspects of Computing 20 (6) (2008) 637662.

[6] Broy, M.: Seamless Model Driven Systems Engineering Based on Formal Models. In *ICFEM 2009*, Lecture Notes in Computer Science 5885 (2009) 1-19.

[7] Calculemus Network. `http://www.calculemus.net`

[8] Coquand, T., Spiwack, A.: Constructively finite? To appear in *Contribuciones científicas en honor de Mirian Andrés Gómez*. Servicio de Publicaciones de la Universidad de La Rioja, 2010.

[9] Coquand, T., Spiwack, A.: Towards Constructive Homological Algebra in Type Theory. In *Calculemus 2007*, Lecture Notes in Artificial Intelligence 4573 (2007) 40-54.

[10] Domínguez, C., Duval, D.: Diagrammatic logic applied to a parameterisation process. Mathematical Structures in Computer Science 20 (4) (2010) 639-654.

[11] Domínguez, C., Lambán, L., Rubio, J.: Object-Oriented Institutions to Specify Symbolic Computation Systems. Rairo - Theoretical Informatics and Applications 41 (2007) 191-214.

[12] Edelsbrunner, H., Harer, J. L.: Computational topology: an introduction. Providence (Rhode Island): American Mathematical Society, 2010.

[13] Jacobson, N.: Basic Algebra II, 2nd edn. W.H. Freeman and Company, 1989.

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

[15] Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A constructive algebraic hierarchy in Coq. Journal of Symbolic Computation 34 (4) (2002) 271-286.

[16] Gonthier, G.: Formal Proof: The Four-Color Theorem. Notices of the AMS 55 (11) (2008) 1382-1393.

[17] Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A Modular Formalisation of Finite Group Theory. In *Theorem Proving in Higher Order Logics 2007*, Lecture Notes in Computer Science 4732 (2007) 86-101.

[18] Lambán, L., Pascual, V., Rubio, J.: An Object-Oriented Interpretation of the EAT System. Applicable Algebra in Engineering, Communication and Computing 14 (3) (2003) 187-215.

[19] Leroy, X.: Formal Verification of a Realistic Compiler. Communications of the ACM 52 (7) (2009) 107-115.

[20] LogiCal project. The Coq Proof Assistant. `http://coq.inria.fr/`, 2010.

[21] Obua, S., Nipkow, T.: Flyspeck II: the basic linear programs. Annals of Mathematics and Artificial Intelligence 56 (2009) 245-272.

[22] Rubio, J., Sergeraert, F.: Homologie effective et suites spectrales d'Eilenberg-Moore. Comptes Rendus de l'Académie Sciences Paris 306 (17) (1988) 723-726.

[23] Rubio, J., Sergeraert, F.: Constructive Algebraic Topology. Bulletin Sciences Mathématiques 126 (2002) 389-412.

[24] Sergeraert, F.: The computability problem in Algebraic Topology. Advances in Mathematics 104 (1994) 1-29.

[25] Spitters, B., van der Weegen, E.: Developing the algebraic hierarchy with type classes in Coq. In *Interactive Theorem Proving 2010*, Lecture Notes in Computer Science 6172 (2010) 490-493.