A certified reduction strategy for homological image processing

María Poza[1], Department of Mathematics and Computer Science, University of La Rioja, Spain
César Domínguez[2], Department of Mathematics and Computer Science, University of La Rioja, Spain
Jónathan Heras[3], School of Computing, University of Dundee, UK
Julio Rubio[4], Department of Mathematics and Computer Science, University of La Rioja, Spain

The *analysis of digital images* using homological procedures is an outstanding topic in the area of Computational Algebraic Topology. In this paper, we describe a *certified* reduction strategy to deal with digital images, but preserving their homological properties. We stress both the advantages of our approach (mainly, the formalisation of the mathematics allowing us to verify the correctness of algorithms) and some limitations (related to the performance of the running systems inside proof assistants). The drawbacks are overcome using techniques that provide an integration of computation and deduction. Our driving application is a problem in bioinformatics, where the accuracy and reliability of computations are specially requested.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic – Mechanical theorem proving; D.2.4 [**Software Engineering**]: Software/Program Verification – Formal methods.

General Terms: Verification, Reliability, Biomedical Imaging

Additional Key Words and Phrases: Homology, Computational Algebraic Topology, Formalization of Mathematics, Coq, SSReflect, Biomedical Images

ACM Reference Format:
María Poza, César Domínguez, Jónathan Heras and Julio Rubio, 2013. A certified reduction strategy for homological image processing. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 22 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Scientific computing is an outstanding tool to assist researchers in experimental sciences. When applied to biomedical problems, the accuracy and reliability of the computations are particularly important. Thus, the possibility of increasing the trust in scientific software by means of mechanized theorem proving technology becomes an interesting area of research.

In this paper, we explore this path to certify image processing procedures. In particular, we have chosen the Coq proof assistant [Bertot and Castéran 2004] to certify the programs which allow us to analyse images obtained from neuron cultures [Cuesto et al. 2011]. The techniques that we use to deal with these images are based on Computational Algebraic Topology. Nowadays, computing in Algebraic Topology has an increasing importance in applied mathematics [Edelsbrunner and Harer 2010]; namely, in the context of digital im-

The work was partially supported by Ministerio de Educación y Ciencia project MTM2009-13842-C02-01[1−4], by the European Union's 7th Framework Programme under grant agreement nr. 243847[1−4], and EPSRC grants EP/J014222/1[3] and EP/K031864/1[3].
Author's addresses: María Poza, César Domínguez and Julio Rubio, Department of Mathematics and Computer Science, University of La Rioja, Spain; J. Heras, School of Computing, University of Dundee, UK

DOI:http://dx.doi.org/10.1145/0000000.0000000

age processing (see [Ayala et al. 2003] and the series of conferences called *Computational Topology in Image Context*). The key observation of our approach is that, after a suitable preprocessing, the solution of a biological problem (namely, the number of synapsis in a picture of a neuron) can be identified with the computation of a topological invariant (the rank of a homology group). Then, all our efforts are concentrated on computing, in a certified manner, such an invariant.

Since the size of real-life biomedical images is too big to handle them directly, we propose a *reduction strategy*, that allows us to work with smaller data structures, but preserving all their homological properties. To this aim, we use the notion of *discrete vector field* [Forman 1998], following very closely an algorithm due to Romero and Sergeraert [Romero and Sergeraert 2010].

In order to verify the correctness of these procedures, it is necessary the formalisation of a certain amount of mathematics. The most significant piece of mathematics formalised in this paper is the so-called *Basic Perturbation Lemma* (or BPL, in short). The proof of this theorem has been already implemented in the Isabelle/HOL proof assistant [Aransay et al. 2008]. The BPL formalisation presented in this paper is much shorter and compact than that of [Aransay et al. 2008]. There are two reasons for this improvement of the formal proof. The former is that we have followed a new and shorter proof of the BPL (due again to Romero and Sergeraert [Romero and Sergeraert 2012]). The latter is that we have built our formal proof on the powerful *SSReflect* library [Gonthier and Mahboubi 2010] of Coq (on the contrary, much of the infrastructure required was defined from scratch in [Aransay et al. 2008]).

Apart from the efficiency in the writing of proofs, using SSReflect also has other consequences. Since SSReflect is designed to deal only with *finite structures*, the proof of the BPL presented here only applies over *finitely generated groups* (the proof formalised in [Aransay et al. 2008] does not have this limitation). Furthermore, dealing with finite structures, and inside the constructive logic of Coq, eases the executability of the proofs, and thus the generation of certified programs (the same tasks in Isabelle/HOL pose more difficulties, see [Aransay et al. 2010]). However, it is worth mentioning that this limitation does not mean any special hindrance in our work, because digital images are always finite structures.

In order to prove the correctness of the generated programs, we must establish, and keep, a link among the initial biomedical picture, and the final smaller data structure where the homological calculations are carried out. This implies a big amount of processing, and does not allow us to execute all the steps *inside* Coq (the full path has been travelled, but only in *toy* examples). Then we have appealed to a programming language, Haskell [Hutton 2007] in our case, to integrate computation and deduction.

Haskell appears in two different steps of our methodology. In the early stages of development, Haskell prototypes of the algorithms are systematically tested by using the QuickCheck tool [Claessen and Hughes 2000]. This allows us to discharge many small and common errors, which could hinder the proving process in Coq. In the final computational step, Haskell is used as an *oracle* for Coq. The most hard parts of the calculation (in our case, an important bottleneck is computing inverse matrices) are delegated to Haskell programs; the *results* of these Haskell programs are then *proved* correct within Coq.

With this hybrid technique, we have got the objective of computing, in a certified way, the homology of actual biomedical images coming from neurological experiments.

The rest of this paper is organized as follows. Section 2 is devoted to present a running example, coming from the biomedical context, as a test-case for our development. The formalisation of an algorithm to build a discrete vector field associated with a matrix, using SSReflect, is explained in Section 3. This vector field computation will be used in Section 4 to reduce the chain complex associated with a digital image. The reduction process is based on an essential lemma in Algorithmic Homological Algebra called the Basic Perturbation Lemma. We also include in that section a proof of such a lemma. In Section 5, we explain

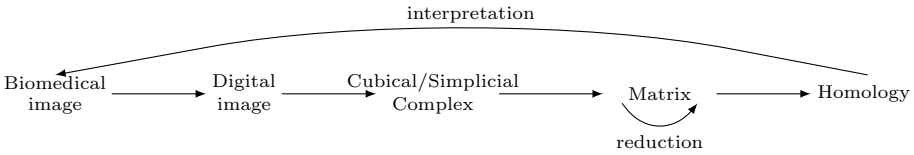

Fig. 1. Computing homology from a digital image.

how the certified programs can be used to effectively compute the homology of images. The paper ends with a section of conclusions and further work, and the bibliography.

The complete source code of our formalisation and the Haskell programs can be seen at http://www.unirioja.es/cu/cedomin/crship/.

2. CERTIFIED IMAGE PROCESSING

The discipline of Algebraic Digital Topology, or more specifically, the computation of homology groups from digital images is mature enough (see, for instance, [Ziou and Allili 2002], one among many good references) to go one step further and investigate the possibility of *certified computations* (i.e., computation formally verified by proving its correctness using an *interactive* proof assistant) in digital topology, as it happens in other areas of computer mathematics (see [Gonthier 2008]).

In a very rough manner, the process to be verified is reflected in Figure 1. Putting it into words, we firstly pre-process a biomedical image to obtain a monochromatic image. From the black pixels of such a monochromatic image a cubical/simplicial complex is obtained (by means of a triangulation procedure); subsequently, from the cubical/simplicial complex, its *boundary (or incidence) matrices* are constructed, and finally, *homology* can be computed. If we work with coefficients over a field (and it is well-known that it is enough to take as coefficients the field $\mathbb{Z}_2$, when working with 2D and 3D digital images [Ayala et al. 2003]) and if only the *degrees* of the homology groups (as vector spaces) are looked for, then having a program able to compute the rank of a matrix is sufficient to accomplish the whole task. In this process, the matrix obtained from the image can be huge. In this case, a process of reduction of the matrix without losing the homological properties of the image can be applied first to compute the homology. In this paper, we particularise this architecture with a real problem that appeared in a biomedical application and with the Coq proof assistant and its SSReflect library as programming and verifying tool.

Biomedical images are a suitable benchmark for testing our programs, the reason is twofold. First of all, the amount of information included in this kind of images is usually quite big. Then, a process able to reduce those images but keeping the homological properties can be really useful. Secondly, software systems dealing with biomedical images must be trustworthy. This is our case since we have formally verified the correctness of our programs.

As an example, we can consider the problem of counting the number of *synapses* in a neuron. Synapses [Bear et al. 2006] are the points of connection between neurons and are related to the computational capabilities of the brain. The possibility of changing the number of synapses may be an important asset in the treatment of neurological diseases, such as Alzheimer, see [Selkoe 2002]. Therefore, we can claim that an efficient, reliable and automatic method to count synapses is instrumental in the study of the evolution of synapses in scientific experiments.

Up to now, the study of the *synaptic density evolution* of neurons was a time-consuming task since it was performed, mainly, manually. To overcome this issue, an automatic method was presented in [Heras et al. 2011]. Briefly speaking, such a process can be split into two parts. Firstly, from three images of a neuron (the neuron with two antibody markers and the structure of the neuron), we obtain a monochromatic image, see Figure 2. In such an image, each connected component represents a synapse. So, the problem of measuring the

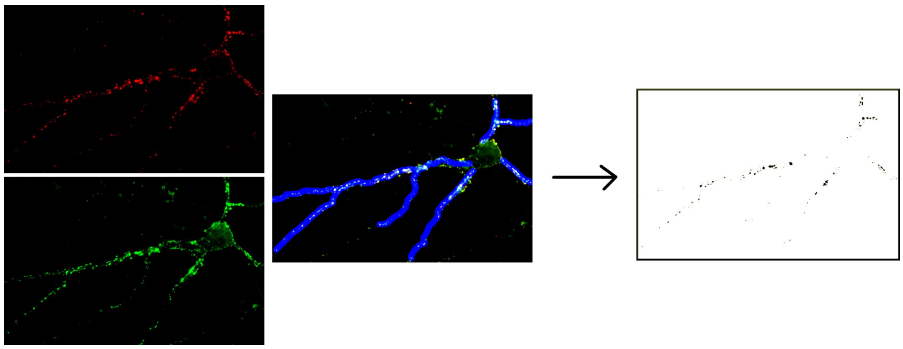

Fig. 2. Synapses extraction from three images of a neuron. The same images with higher resolution can be seen in `http://www.unirioja.es/cu/joheras/synapses/`.

number of synapses is translated into a question of counting the connected components of a monochromatic image.

In the context of Algebraic Digital Topology, this issue can be tackled by means of the computation of the homology group H_0 of the monochromatic image. This task can be performed in Coq through the formally verified programs presented in [Heras et al. 2012a]. Nevertheless, such programs are not able to handle images like the one of the right side of Figure 2 due to its size (the images contain up to 10^6 pixels). It is worth noting that Coq is a Proof Assistant and not a Computer Algebra system; and, in general, efficiency of algorithms is pushed into the background of this kind of systems. However, there is an effort towards the efficient implementations of mathematical algorithms running inside Coq, as shown by recent works on efficient real numbers [Krebbers and Spitters 2011], machine integers and arrays [Armand et al. 2010] or an approach to compiled execution of internal computations [Grégoire and Leroy 2002].

In our case, we apply a reduction process of the data structures but without loosing the homological properties to overcome the efficiency problem. In particular, we are focused on the formalisation of *discrete vector fields*, a powerful notion that has been welcomed in the study of homological properties of digital images, see [Cazals et al. 2003; Gyulassy et al. 2008; Jerse and Kosta 2010]. The importance of discrete vector fields, which were first introduced in [Forman 1998], stems from the fact that they can be used to considerably reduce the amount of information of a discrete object but preserving homological properties. Using this approach, we can successfully compute the homology of the previous biomedical image in just 25 seconds, a remarkable time for an execution inside Coq. Besides, we have proved using this proof assistant that the homological properties of the initial digital image and the reduced one are preserved.

3. DISCRETE VECTOR FIELDS

In this section, we include the basic definitions which, mainly, come from the algebraic setting of Discrete Morse Theory presented in [Romero and Sergeraert 2010]. In addition, we present an algorithm to construct an admissible discrete vector field from a matrix. Then, a formalisation in Coq of this algorithm is provided. Finally, we introduce the fundamental notions in the Effective Homology theory [Rubio and Sergeraert 2002] and state the theorem where Discrete Morse Theory and Effective Homology converge.

3.1. Basic mathematical definitions

We assume as known the notions of *ring*, *module* over a ring and *module morphism* (see, for instance, [Jacobson 1989]). First of all, let us introduce one of the main notions in the context of Algebraic Topology: *chain complexes*.

DEFINITION 1. *A* chain complex C_* *is a pair* (C, d)*, where* $C = \{C_n\}_{n\in\mathbb{Z}}$ *is a family of* $\mathcal{R}$*-modules and* $d = \{d_n : C_n \to C_{n-1}\}_{n\in\mathbb{Z}}$ *is family of module morphisms, called the* differential map*, such that* $d_{n-1} \circ d_n = 0$*, for all* $n \in \mathbb{Z}$*. In many situations the ring* $\mathcal{R}$ *is either the integer ring,* $\mathcal{R} = \mathbb{Z}$*, or the field* $\mathbb{Z}_2$*. Usually, we denote the chain complex* $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$*. A chain complex is* free *(*of finite type*) if its modules are free (finitely generated).*

The image $B_n = im\ d_{n+1} \subseteq C_n$ is the (sub)module of *n-boundaries*. The kernel $Z_n = \ker d_n \subseteq C_n$ is the (sub)module of *n-cycles*. Given a chain complex $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$, the identities $d_{n-1} \circ d_n = 0$ mean the inclusion relations $B_n \subseteq Z_n$: every boundary is a cycle (the converse in general is not true). Thus the next definition makes sense.

DEFINITION 2. *The* n-homology group *of* C_**, denoted by* $H_n(C_*)$*, is defined as the quotient* $H_n(C_*) = Z_n/B_n$*.*

Chain complexes have a corresponding notion of morphism.

DEFINITION 3. *Let* $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ *and* $D_* = (D_n, \widehat{d}_n)_{n\in\mathbb{Z}}$ *be two chain complexes. A* chain complex morphism $f : C_* \to D_*$ *is a family of module morphisms,* $f = \{f_n : C_n \to D_n\}_{n\in\mathbb{Z}}$*, satisfying the relation* $f_{n-1} \circ d_n = \widehat{d}_n \circ f_n$*, for all* $n \in \mathbb{Z}$*. Usually, the sub-indexes are skipped, and we just write* $f \circ d = \widehat{d} \circ f$*.*

Let us state now the main notions coming from the algebraic setting of Discrete Morse Theory [Romero and Sergeraert 2010].

DEFINITION 4. *Let* $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ *be a free chain complex with a distinguished* $\mathbb{Z}$*-basis* $\beta_n \subset C_n$*, for all* $n \in \mathbb{Z}$*; every basis component* $\sigma \in \beta_n$ *is an* n*-cell or simply a cell. A* discrete vector field V *on* C_* *is a collection of pairs* $V = \{(\sigma_i, \tau_i)\}_{i\in I}$ *(where* I *is a finite set) satisfying the conditions:*

- *Every* σ_i *is some element of* β_n*, in which case* $\tau_i \in \beta_{n+1}$*. The degree* n *depends on* i *and in general is not constant.*
- *Every component* σ_i *is a* regular face *of the corresponding* τ_i *(regular face means that the coefficient of* σ_i *in* $d_{n+1}(\tau_i)$ *is* 1 *or* -1*).*
- *Each cell of* C_* *appears at most one time in any of the two components of a pair in* V*.*

It is not compulsory that all the cells of C_* appear in the vector field V.

DEFINITION 5. *A cell* χ *which does not appear in a discrete vector field* $V = \{(\sigma_i, \tau_i)\}_{i\in I}$ *is called a* critical cell*. The elements* σ_i *and* τ_i *in the vector field are called* source *and* target cells*, respectively.*

From a discrete vector field V on a chain complex, we can introduce the notion of V-paths.

DEFINITION 6. *A* V*-path of degree* n *and length* m *is a sequence* $((\sigma_{i_k}, \tau_{i_k}))_{0\leq k<m}$ *satisfying:*

- *Every pair* $(\sigma_{i_k}, \tau_{i_k})$ *is a component of* V *and* τ_{i_k} *is an* n*-cell.*
- *For every* $0 < k < m$*, the component* σ_{i_k} *is a face of* $\tau_{i_{k-1}}$ *(the coefficient of* σ_{i_k} *in* $d_n(\tau_{i_{k-1}})$ *is non-null) different from* $\sigma_{i_{k-1}}$*.*

DEFINITION 7. *A discrete vector field* V *is* admissible *if for every* $n \in \mathbb{Z}$*, a function* $\lambda_n : \beta_n \to \mathbb{N}$ *is provided satisfying the following property: every* V*-path starting from* $\sigma \in \beta_n$ *has a length bounded by* $\lambda_n(\sigma)$*.*

In this way, infinite paths are avoided in an admissible discrete vector field. This condition will play a key role in the proof of one of the most important results of our formalisation: the Vector Field Reduction Theorem, see Theorem 13.

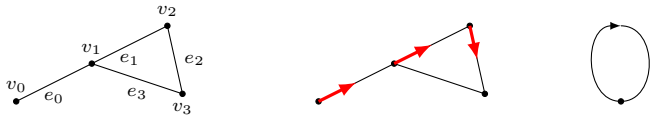

Fig. 3. A chain complex, an admissible discrete vector field on it, and the reduction.

An admissible discrete vector field provides a decomposition of the generators of the chain complex into "useless" elements (in the sense, that they can be removed without changing its homology) and *critical* elements (those whose removal could modify the homology).

EXAMPLE 8. *Let us consider the free chain complex C_* represented in the left side of Figure 3. The 0-cells of C_* are the vertices v_0, v_1, v_2 and v_3; and the 1-cells are the edges e_0, e_1, e_2 and e_3, there is not any other cell in the chain complex. The differential map of C_* for the 0-cells is the null map and the differential map for the 1-cells is defined as $d_1(e_i) = 1 \times e_{i+1} - 1 \times e_i$ with $i \in \{0, 1, 2, 3\}$. The admissible discrete vector field $V = \{(v_0, e_0), (v_1, e_1), (v_2, e_2)\}$ of C_* is represented in the centre of Figure 3. In this case, the critical cells are v_3 and e_3.*

If we consider the case of *finite type* chain complexes, where there is a finite number of generators in each degree (as in the previous example), the differential maps can be represented as matrices.

DEFINITION 9. *Let $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ be a finite type chain complex with a distinguished $\mathbb{Z}$-basis $\beta_n \subset C_n$, for all $n \in \mathbb{Z}$. Assuming an ordering on the generators of the same dimension and $\forall n \in \mathbb{Z}$, we define the* differential matrix *associated with d_n as an $m1 \times m2$ matrix M_n (where $m1$ and $m2$ are respectively the number of elements of β_{n-1} and β_n) such that the element of position (i, j) of M_n is the value of the i-th element of β_{n-1} when applying d_n to the j-th element of β_n.*

When working with matrices, the concept of admissible discrete vector field can be reworded in the following way (these are the definitions used in our actual formalisation).

DEFINITION 10. *Let M be a matrix with coefficients in $\mathbb{Z}$, and with m rows and n columns. A* discrete vector field *V for M is a set of pairs of natural numbers $\{(a_i, b_i)\}_{i\in I}$ satisfying, for all $i \in I$, the following conditions:*

- $1 \leq a_i \leq m$ *and* $1 \leq b_i \leq n$.
- $M[a_i, b_i] = \pm 1$.
- *The indexes a_i (resp. b_i) are pairwise different.*

Given two source cells a and a' of V such that $a \neq a'$, we say that $a > a'$ if a vector (a, b) is present in V and $M[a', b]$ is non-null (i.e. if there is an elementary V-path from a to a'). A discrete vector field V for M is admissible *if this binary relation on source cells transitively generates a partial order (i.e. if there is no loop $a_1 > a_2 > \ldots > a_k = a_1$).*

3.2. Romero-Sergeraert's algorithm

All the algorithms devoted to construct admissible discrete vector fields share the same goal: the construction of an admissible discrete vector field as big as possible (see for instance [Kozlov 2007] or [Lewiner et al. 2004]). Some of them return a vector field quickly, but others spend more time to compute it. Let us emphasize that the latter ones are notable because the search has been more thorough, so the number of vectors will be higher. We are interested in an algorithm which not only gives us a big vector field but also does not spend too much time to compute it. The Romero-Sergeraert algorithm (from now on RS algorithm) does not always build the biggest vector field possible, but the number of vectors

is quite close to the biggest one. Furthermore, in many cases, it returns the best possible vector field. Moreover, it is fast enough to obtain the vector field in our application domain. Due to these reasons, the RS algorithm has been chosen to make our computations.

Briefly, the RS algorithm builds an admissible discrete vector field by running through the rows of a matrix. It looks for the first element in the row which verifies the admissibility property with respect to the elements previously included in the discrete vector filed. We define the RS algorithm as follows.

ALGORITHM 11 (THE RS ALGORITHM).
Input: a matrix M with coefficients in $\mathbb{Z}$.
Output: an admissible discrete vector field V for M and a list of relations r between row indexes.
Description:

(1) *Initialise the vector field V to the void vector field and the relations r to empty.*
(2) *For every row i of M:*
  2.1. *For every column j, which is different from the second components of V, such that $M[i,j] = 1$ or $M[i,j] = -1$:*
    Look for the rows $k \neq i$ such as $M[k,j] \neq 0$ and obtain the relations $i > k$. Then, build the transitive closure of r and these relations.
      `If` *there is no loop in that transitive closure:*
        `then`*: Add (i,j) to V, let r be that transitive closure, and repeat from Step 2 with the next row.*
        `else`*: Repeat from Step 2.1. with the next column.*

In general, this algorithm can be applied over matrices with coefficients in a ring. In that case, the condition $M[i,j] = 1 \vee M[i,j] = -1$ will be replaced by $M[i,j]$ is a unit of the ring. Specifically, if we work with a field F, instead of a ring, every non-null element is a unit. In our particular case, as the homology groups of 2D images are torsion-free, we will work with the field $\mathbb{Z}_2$. Therefore, the selected vectors are entries whose value is 1. From now on, we will work with $\mathbb{Z}_2$.

Let us mention that it is relevant sorting the admissible discrete vector field because we will sort the matrix prior to reducing it. For every vector (a,b), the value of the function $\lambda(a)$, which gives us the longest path from a, is computed. In our case, as we build the transitive closure, it is the maximum length of the relations which start with a. Then, we sort the vector field by the values of λ in decreasing order. If we have two equal values of λ, the chosen order is not relevant. Then, the rows and columns of the matrix are sorted using the ordered vector field. This reordered matrix will have a lower triangular matrix with 1's in the diagonal as upper-left submatrix of dimension the number of vectors in the vector field.

3.3. Formalisation of the RS algorithm in SSReflect

The development of a formally certified implementation of the RS algorithm was presented in [Heras et al. 2012b]. It followed the methodology presented in [Mörtberg 2010]. Firstly, the programs were implemented in Haskell [Hutton 2007], a *lazy* functional programming language. Subsequently, our implementation was intensively tested using QuickCheck [Claessen and Hughes 2000], a tool which automatically tests properties about programs implemented in Haskell. Finally, the correctness of our programs was verified using the Coq [Bertot and Castéran 2004] proof assistant and its SSReflect library [Gonthier and Mahboubi 2010]. In this section, we briefly show the last step of this process.

First of all, we define the data types related to our programs. A matrix is represented by means of a list of lists of the same length over $\mathbb{Z}_2$ (encoded using the unit ring type associated with the booleans), a vector field is a sequence of natural pairs, and finally, the

relations are a list of lists of natural numbers. The notation `a:T` means that the variable `a` has type `T`, and the function `rowseqmx M i` returns the i-th row of the matrix `M`.

```
Definition Z2:= bool_cunitRingType.
Definition matZ2 := seqmatrix bool_cunitRingType.
Definition is_matrix m n (M:matZ2) := M = [::] \/
    [/\ m = size M & forall i, i < m -> size (rowseqmx M i) = n].
Definition vectorfield:= seq (prod nat nat).
Definition orders:= seq(seq nat).
```

Then, we have defined a function called `Vecfieldadm` (this function is a boolean-valued theorem) that checks if an element `vf: vectorfield` satisfies the properties of an admissible discrete vector field with respect to a matrix `M: matZ2` and a list of relations `r: orders`.

```
Definition Vecfieldadm (M: matZ2)(vf: vectorfield)(r:orders) :=
  (longmn (size M) (getfirstElemseq vf)) /\
  (longmn (size (nth [::] M 0)) (getsndElemseq vf)) /\
  (uniq (getfirstElemseq vf)) /\
  (uniq (getsndElemseq vf)) /\
  (forall i j:nat, (i,j) \in vf -> compij i j M = true) /\
  (forall i j m:nat, (i,j)\in vf -> i!=m -> compij m j M != false -> (i::
      m::nil)\in r) /\
  (forall a b s p, (a::s) \in r -> (last 0%N s = b)
       -> (b::p) \in r -> ((a::s) ++ p) \in r) /\
  (norep r) /\
  (ordered glMax vf r).
```

Let us note that the first five conditions come from the three properties of a discrete vector field (see Definition 10). The next three conditions are linked with the relations. The first one gives us the link between the vector field and the relations. The second one verifies that we are constructing the transitive closure. And the last one states the admissibility property. It makes sure that every sequence of `r` has not repeated elements. Finally, it is checked that `vf` is ordered taking into account the `glMax` function, which returns the longest path associated with a cell, and the relations `r`.

The RS algorithm has been implemented using two functions: `genDvf`, which constructs an admissible discrete vector field, and `genOrders`, which generates the relations. As we have explained in the last paragraph of the previous subsection, the discrete vector field generated by the RS algorithms must be reordered, this is achieved thanks to the function `dvford`. The correctness of this program has been proved in the theorem `dvfordisVecfieldadm` – this theorem establishes that given a matrix `M`, the output produced by `dvford` satisfies the properties specified in `Vecfieldadm`.

```
Variable M: matZ2.
Variable m n: nat.
Hypotheses ismatrix: is_matrix m n M.
Theorem dvfordisVecfieldadm: Vecfieldadm M (dvford M)(genOrders M).
```

The proof of the above theorem has been split into a series of lemmas which correspond to each one of the properties that should be fulfilled to have an admissible discrete vector field. For instance, the lemma associated with the first property of the definition of a discrete vector field is the following one.

```
Lemma propSizef(M:matZ2): (longmn (size M) (getfirstElemseq (genDvf M))).
```

Both the functions which implement the RS algorithm and the ones needed to specify the properties of admissible discrete vector fields are defined using a *functional style*; that

is, our programs are defined using *pattern-matching* and *recursion*. Therefore, in order to reason about our recursive functions, we need elimination principles which are fitted for them. To this aim, we use the tool presented in [Barthe and Courtieu 2002] to reason about complex recursive definitions in Coq.

For instance, in our development of the implementation of the RS algorithm, we have defined a function, called `subm`, which takes as arguments a natural number `n` and a matrix `M` and removes the first `n` rows of `M`. The inductive scheme associated with `subm` is set as follows.

```
Functional Scheme subm_ind := Induction for subm Sort Prop.
```

The statement of the principle generated by this inductive scheme is the following one:

```
forall P : nat -> matZ2 -> matZ2 -> Prop,
(forall (n : nat) (m : matZ2), m = [::] -> P n [::] [::]) ->
 (forall (n : nat) (m : matZ2) (a : seqZ2) (b : matZ2),
  m = a :: b -> n = 0 -> P 0 (a :: b) [::]) ->
  (forall (n : nat) (m : matZ2) (a : seqZ2) (b : matZ2), m = a :: b ->
    forall p : nat, n = p.+1 -> P p b (subm p b) ->
     P p.+1 (a :: b) (a :: subm p b)) -> forall (n : nat) (m : matZ2),
      P n m (subm n m)
```

Then, when we need to reason about `subm`, we can apply this scheme with the corresponding parameters using the instruction `functional induction`. However, as we have previously said both our programs to define the RS algorithm and the ones which specify the properties to prove are recursive. Then, in several cases, it is necessary to merge several inductive schemes to induct simultaneously on several variables. For instance, let M be a matrix and M' be a submatrix of M where we have removed the $(k-1)$ first rows of M; then, we want to prove that $\forall j,\ M(i,j) = M'(i-k+1,j)$. This can be stated in Coq as follows.

```
Lemma Mij_subM (i k: nat) (M: matZ2):
  k <= i -> k != 0 -> let M' := (subm k M) in
  M i j == M' (i - k + 1) j.
```

To prove this lemma it is necessary to induct simultaneously on the parameters `i`, `k`, and `M`, but the inductive scheme generated from `subm` only applies induction on `k` and `M`. Therefore, we have to define a new recursive function, called `Mij_subM_rec`, to provide a proper inductive scheme.

```
Fixpoint Mij_subM_rec (i k: nat) (M: matZ2) :=
match k with
|0 => M
|S p => match M with
      |nil => nil
      |hM::tM => if (k == 1)
                 then a::b
                 else (Mij_subM_rec p (i- 1) tM)
      end
end.
```

Then, the function `Mij_subM_rec` is used as an inductive scheme for the proof of Lemma `Mij_subM`. This style of proving functional programs in Coq is the one followed in the development of the proof of Theorem `dvfordisVecfieldadm`.

3.4. Vector Field Reduction theorem

In this section, we state the theorem where Discrete Morse Theory and Effective Homology converge. First, we introduce *reductions* [Rubio and Sergeraert 1997], one of the fundamental notions in the Effective Homology theory.

DEFINITION 12. *A* reduction ρ *between two chain complexes* $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ *and* $D_* = (D_n, \hat{d}_n)_{n\in\mathbb{Z}}$, *denoted by* $\rho : C_* \Rightarrow D_*$, *is a triple* $\rho = (f, g, h)$ *where* $f : C_* \to D_*$ *and* $g : D_* \to C_*$ *are chain complex morphisms,* $h = \{h_n : C_n \to C_{n+1}\}_{n\in\mathbb{Z}}$ *is a family of module morphism, and the following properties are satisfied:*

1) $f \circ g = id$;
2) $g \circ f + d \circ h + h \circ d = id$;
3) $f \circ h = 0; \quad h \circ g = 0; \quad h \circ h = 0.$

The importance of reductions lies in the fact that given a reduction $\rho : C_* \Rightarrow D_*$, then $H_n(C_*)$ is isomorphic to $H_n(D_*)$ for every $n \in \mathbb{Z}$. Very frequently, D_* is a much smaller chain complex than C_*, so we can compute the homology groups of C_* much faster by means of those of D_*.

THEOREM 13 (VECTOR FIELD REDUCTION THEOREM). *Let* $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ *be a free chain complex and* V *be an admissible discrete vector field on* C_*. *Then, the vector field* V *defines a canonical reduction* $\rho : (C_n, d_n)_{n\in\mathbb{Z}} \Rightarrow (C^c_n, d^c_n)_{n\in\mathbb{Z}}$ *where* $C^c_n = \mathbb{Z}[\beta^c_n]$ *is the free* $\mathbb{Z}$*-module generated by* β^c_n, *the critical* n*-cells of* C_*.

Therefore, the bigger the admissible discrete vector field V the smaller the chain complex C^c_*.

A quite direct proof of the Vector Field Reduction Theorem based on the Basic Perturbation Lemma appeared in [Romero and Sergeraert 2010].

THEOREM 14 (BASIC PERTURBATION LEMMA, BPL). *Let* $\rho = (f, g, h) : (C, d) \Rightarrow (\widehat{C}, \widehat{d})$ *be a reduction, and* δ *be a* perturbation *of* d, *that is,* δ *is a morphism such that* $d + \delta$ *is a differential map for* C. *Furthermore, the function* $\delta \circ h$ *is assumed* locally nilpotent, *in other words, for every* $x \in C$ *there exists* $m \in \mathbb{N}$ *(in general,* m *can depend on* x*) such that* $(\delta \circ h)^m(x) = 0$. *Then, a perturbation* $\widehat{\delta}$ *can be defined for the differential* $\widehat{d}$ *and a new reduction* $\rho' : (C, d + \delta) \Rightarrow (\widehat{C}, \widehat{d} + \widehat{\delta})$ *can be constructed.*

The proof of the Vector Field Reduction theorem based on the BPL is quite simple. Let us explain this proof in a nutshell. First of all, we consider a particular chain complex (C, ϱ) with the same underlying graded module than (C, d), but with a simplified differential map. Namely, each component $\varrho_n : C_n \to C_{n-1}$ is defined in the following way. It is clear that the vector field V defines a canonical decomposition of C_n depending on the generators are source, target, or critical cells. Then, ϱ_n apply to 0 the source and critical cells. If τ is a target cell, there is a unique vector $(\sigma, \tau) \in V$. Then, $\varrho_n(\tau) = \varepsilon(\sigma, \tau)\sigma$, where $\varepsilon(\sigma, \tau)$ is the coefficient 1 or -1 of σ in $d_n(\tau)$. A homotopy operator h is defined in the same way as ϱ but in the reverse direction (i.e. $h_n(\sigma) = \varepsilon(\sigma, \tau)\tau$).

Then, we can define an initial reduction, $\rho = (f, g, h) : (C, \varrho) \Rightarrow (C^c, 0)$, of the previous chain complex to the chain complex which modules are generated by the critical cells and which differential map is null. Now, let us define the morphism $P = (d - \varrho)$ which is clearly a perturbation of ϱ. If the nilpotency condition is satisfied, then the BPL produces the required reduction. The composition $(d - \varrho) \circ h$ is not null only for source cells. For a source cell σ, the images $((d - \varrho) \circ h)^m(\sigma)$ correspond to walking V-paths starting at this cell. As the vector field is admissible, the length of these paths is bounded, the image goes eventually to zero, and the nilpotency condition is obtained.

Again, in the case of finite type chain complexes which differentials are represented as matrices, given an admissible discrete vector field for those matrices, we can construct new matrices taking into account the critical components. These smaller matrices define chain complexes which preserve the homological properties. This is the equivalent version of Theorem 13 for finite type chain complexes. A detailed description of the process can be seen in [Romero and Sergeraert 2010].

4. BASIC PERTURBATION LEMMA

In this section, we introduce a formalisation in SSReflect of the BPL, a central lemma in Algorithmic Homological Algebra – in particular, it has been intensively used in the Kenzo Computer Algebra system [Dousson et al. 1998]. In the literature, there are several ways of proving this lemma (see, for instance [Barnes and Lambe 1991; Rubio and Sergeraert 1997]). There are also works related to the formalisation of the BPL. For instance, the non-graded case of this lemma is proved in *Isabelle/HOL* [Aransay et al. 2008]. Furthermore, a particular case of the BPL is also proved in *Coq* using bicomplexes [Domínguez and Rubio 2011]. Now, we show a formalisation of the general case in SSReflect but with finite type structures. Let us recall that SSReflect only works with finite types; so, it can seem that this technology can restrict our development. Nevertheless, this approach is enough because we are interested in applying the BPL to the computation of the homology associated with a digital image – a case where we have a finite type chain complex. Indeed, in this context, this lemma is applied to a reduction where most of the differentials are null.

4.1. Mathematical proof of the BPL

The proof of the BPL (explained in [Romero and Sergeraert 2012]) is based on two results. The former, named Decomposition Theorem, builds a decomposition of a chain complex from a reduction of it. The latter is a Generalisation of the Hexagonal Lemma.

THEOREM 15 (DECOMPOSITION THEOREM). *Let $\rho = (f, g, h) : (C, d) \Rightarrow (\widehat{C}, \hat{d})$ be a reduction. This reduction can be used to obtain a decomposition $C_n = A_n \oplus B_n \oplus C'_n$ where: $C'_n = im\ g_n$, $A_n \oplus B_n = \ker f_n$, $A_n = \ker f_n \cap \ker h_n$, $B_n = \ker f_n \cap \ker d_n$, the module morphisms f_n and g_n induce isomorphisms between C'_n and $\widehat{C}_n$, and the arrows d_n and h_{n-1} induce module isomorphisms between A_n and B_{n-1}.*

These properties are illustrated in the diagram represented in Figure 4. It is a simple exercise of elementary linear algebra to prove the equivalence between the diagram of Figure 4 and the initial reduction.

The Hexagonal Lemma [Romero and Sergeraert 2010] allows us to reduce only a module of a chain complex in a particular degree. It is possible to generalize this lemma applying the reduction to every degree simultaneously [Romero and Sergeraert 2010].

THEOREM 16 (GENERALISATION OF THE HEXAGONAL LEMMA). *Let $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ be a chain complex. We assume that every module is decomposed $C_n = A_n \oplus B_n \oplus C'_n$. The differential map d_n are then decomposed in 3×3 block matrices $[d_{n,i,j}]_{1\leq i,j\leq 3}$. If every component $d_{n,2,1} : A_n \to B_{n-1}$ is an isomorphism, then the chain complex can be canonically reduced to a chain complex (C'_n, d'_n). The components of the desired reduction are:*

$$d'_n = d_{n,3,3} - d_{n,3,1}d_{n,2,1}^{-1}d_{n,2,3} \qquad f_n = \begin{bmatrix} 0 & -d_{n,3,1}d_{n,2,1}^{-1} & 1 \end{bmatrix}$$

$$g_n = \begin{bmatrix} -d_{n,2,1}^{-1}d_{n,2,3} \\ 0 \\ 1 \end{bmatrix} \qquad h_{n-1} = \begin{bmatrix} 0 & d_{n,2,1}^{-1} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

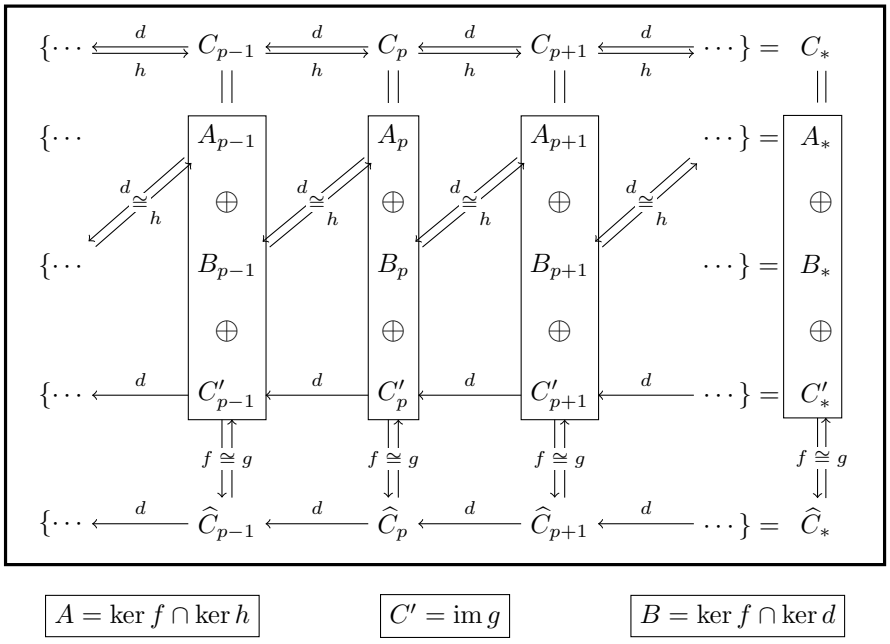

Fig. 4. Decomposition Theorem diagram.

Again, it is not difficult to check that the displayed formulas satisfy the relations of a reduction stated in Definition 12.

With these two auxiliary lemmas, it is possible to obtain a proof of the BPL. The process is the following one. Given a reduction $\rho = (f, g, h) : (C, d) \Rightarrow (\widehat{C}, \widehat{d})$ and a perturbation δ of the differential d such that $\delta \circ h$ is locally nilpotent, it is necessary to build a new reduction $\rho' = (f', g', h') : (C, d + \delta) \Rightarrow (\widehat{C}, \widehat{d} + \widehat{\delta})$.

The reduction $\rho : C_* \Rightarrow \widehat{C}_*$ allows us to apply the Decomposition Theorem and to obtain the diagram of Figure 4 where $C = A \oplus B \oplus C'$. Then, the differential $(d+\delta)$ can be depicted in nine blocks following that decomposition. If the component $(d+\delta)_{21} : A \to B$ is invertible then the Generalisation of Hexagonal Lemma can be applied and the BPL is proved.

We have that $(d + \delta)_{21} = d_{21} + \delta_{21} = d_{21} + \delta_{21} \circ h_{12} \circ d_{21} = (id + \delta_{21} \circ h_{12}) \circ d_{21}$. Then, as d_{21} is invertible (in fact, h_{12} is its inverse), we focus on proving that the other member of the product, namely $(id + \delta_{21} \circ h_{12})$, is invertible.

As $\delta \circ h$ is locally nilpotent, i.e. for every $x \in C$, there exists $m \in \mathbb{N}$ satisfying $(\delta \circ h)^m(x) = 0$, we obtain that in particular $(\delta_{21} \circ h_{12})^m(x) = 0$ since the unique non-null component of h is h_{12}. Then, the inverse of $(id + \delta_{21} \circ h_{12})$ is $\sum_{i=1}^{\infty} (-1)^i (\delta_{21} \circ h_{12})^i$.

4.2. Formalisation of the BPL

The formalisation of the proof of the BPL in SSReflect requires to restrict the data structures to finite type chain complex. These structures are presented in Subsection 4.2.2. Then, the Decomposition Lemma and the Generalized Hexagonal Lemma are formalised in Subsection 4.2.3 and 4.2.4. These two results are the key ingredients used in the formalisation of the BPL – included in Subsection 4.2.5. Before that, we start providing a brief explanation about the formalisation of the kernel of a map in SSReflect. The representation chosen for this well-known notion in mathematics has important consequences in the rest of structures used in the formalisation of the proof.

4.2.1. The kernel of a map. The kernel of a finite map is defined by the kernel of the matrix which represents this map. This is defined in SSReflect in the following way.

```
Definition kermx m n (A: 'M_(m,n)): 'M_m :=
  copid_mx (\rank A) *m invmx (col_ebase A).
```

The kernel of a matrix `A` is defined as the inverse of `col_ebase` (the extended column basis of `A`), with the top `\rank A` rows zeroed out – this is achieved thanks to the multiplication of such inverse by a square diagonal matrix with 1's on all but the first `\rank A` coefficients on its main diagonal (`copid_mx (\rank A)`). In other words, the kernel of a matrix A is a square matrix whose row space consists of some `u` such that `u *m A = 0`. This property is expressed in the following lemma.

```
Lemma mulmx_ker m n (A : 'M_(m, n)) : kermx A *m A = 0.
```

Two comments are required about this definition. Firstly, the kernel consists of the elements that are made null when they are applied to the left. In our mathematical proofs of Section 4.1 the kernel consists of the elements that are made null when applied to the right. Secondly, in our Coq development, we have chosen to delete the top `\rank A` null rows of the kernel `kermx A`. If we worked with the original definition of `kermx A`, we would obtain partial identities instead of identities, for instance, in the proofs of the equalities in Theorem 15.

```
Definition ker_min (m n : nat) (M : 'M_(m,n)) :=
  (castmx ((Logic.eq_refl (m-\rank M)),(\rank M) + (m-(\rank M))) = m)
  (row_mx (@const_mx _ (m-\rank M) (\rank M) 0) 1%:M)) *m (kermx M).
```

The previous definition consists of the product of a row matrix with the kernel. The row matrix is composed by a block of zeros with `m-\rank M` rows and `\rank M` columns and an identity matrix of dimension `\rank M`. Let us note that the cast (a *coercion* which allows us to change an entity of one data type into another) in the definition is necessary so that the product can be properly defined. In particular, the type `\rank M + m-\rank M` does not reduce to `m`, but is just provably equal. Anyway, both definitions, `kermx M` and `ker_min M`, generate the same space as we can see in the following lemma.

```
Lemma ker_min_kermx (m n : nat) (M : 'M_(m,n)) :
  (kermx M :=: (ker_min M))%MS.
```

4.2.2. Main mathematical structures. Let us define a finite type chain complex with elements in a field.

```
Variable K: fieldType.
Record FGChain_Complex :=
 {m: Z -> nat;
  diff: forall i: Z, 'M[K]_(m (i + 1), m i);
  boundary: forall i: Z, (diff (i + 1)) *m (diff i) = 0}.
```

Some comments about this definition are necessary. The chain complex definition contains a function denoted by `m` which obtains the number of generators for each degree. Then, we can define the differentials using the matrix representation of these maps `forall i: Z , 'M[K]_(m (i + 1), m i)`. Two important design decisions have been included in this definition. Due to the definition of the kernel of a matrix in SSReflect we will work with transposed matrices. This implies that the product is also reversed. Furthermore, the degrees of the differentials have been increased in one unit. This is an alternative definition to the usual one which considers the differential in degree `i` as a function from degree `i+1` to degree `i` [Domínguez and Rubio 2011]. It is clear that as we are considering the definition for all

the integers, both definitions are equivalent. However, with our version, a Coq technical problem, related to indexes, is easily avoided – the concrete problem will be described in the following paragraphs.

Now, we can define the notions of chain complex morphism and homotopy operator for the `FGChain_Complex` structure.

```
Record FGChain_Complex_Morphism (A B: FGChain_Complex) :=
{FG : forall i: Z, 'M[K]_(m A i, m B i);
 FG_well_defined: forall i: Z, (diff A i) *m (FG i) =
                               (FG (i+1)) *m (diff B i)}.

Record FGHomotopy_operator (A: FGChain_Complex) :=
 {Ho: forall i:Z, 'M[K]_(m A i, m A (i+1)%Z)}.
```

With these previous structures, we can define the notion of a reduction for a finite type chain complex.

```
Record FGReduction :=
  {C : FGChain_Complex;
   D : FGChain_Complex;
   F : FGChain_Complex_Morphism C D;
   G : FGChain_Complex_Morphism D C;
   H : FGHomotopy_operator C;
   ax1 : forall i:Z, (M G i) *m (M F i) = 1%:M;
   ax2 : forall i:Z, (M F (i+1)) *m (M G (i+1)) +
                     ((Ho H (i+1)) *m (diff C (i+1))) +
                     ((diff C i) *m (Ho H i)) = 1%:M;
   ax3 : forall i:Z, (Ho H i) *m (M F (i+1)) = 0;
   ax4 : forall i:Z, (M G i) *m (Ho H i) = 0;
   ax5 : forall i:Z, (Ho H i) *m (Ho H (i+1)) = 0}.
```

We are going to focus our attention on the `(diff C i)*m (Ho H i)` component of the definition of a reduction. This product is possible without casts. If we consider the mathematically equivalent definition with the differential in degree `i` as a matrix `'M_(m i,m(i-1))`, then the corresponding component would be `(diff C i)*m (Ho H (i-1))`. In this case a cast would be required to transform elements in degree `(i-1+1)` into elements in degree `i`. Using such a definition this kind of casts would populate the development; however, using our alternative definition, we avoid the use of casts.

4.2.3. Formalisation of the Decomposition Theorem. In this subsection, we focus on proving the Decomposition Theorem. The powerful SSReflect library on matrices makes this development easier. Given a reduction `rho: FGReduction K` we define the decomposition of `(C rho)` through an isomorphism between the module with `(m (C rho)i)` generators and the module built from the sum of three set of generators. The first morphism of the isomorphism reflects a change of basis between both modules (where we work with the canonical basis in the first one) taking into account how the second one is divided:

```
Definition Fi_isom (i: Z):=
 (col_mx (ker_min (row_mx (FG (F rho) (i+1)) (Ho (H rho) (i+1))))
        (col_mx (ker_min (row_mx (FG (F rho) (i+1)) (diff (C rho) i)))
          (FG (G rho) (i+1)))).
```

In this definition, it is necessary to note that the row space of the column matrix `col_mx A B` is the sum of the row spaces of the matrices `A` and `B`, and that the intersection of kernels of two matrices `A` and `B` generates the same space that the kernel of a row matrix `row_mx A B`.

The definition of the inverse morphism `Gi_isom` is obtained knowing that it is a row matrix composed by three blocks and using the second property of the reduction definition. Then, applying the change of basis to the source chain complex of the reduction `(C rho)` we obtain a new reduction between the decomposed chain complex and the second chain complex of the reduction `(D rho)`. It is not difficult to prove that the components of that reduction are:

$$\texttt{D\_rho\_base i} = \begin{pmatrix} 0 & \texttt{D\_aux i} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \texttt{diff (D rho)(i+1)} \end{pmatrix} \quad \texttt{F\_rho\_base i} = \begin{pmatrix} 0 \\ 0 \\ \texttt{id} \end{pmatrix}$$

$$\texttt{G\_rho\_base i} = \begin{pmatrix} 0 & 0 & \texttt{id} \end{pmatrix} \quad \texttt{H\_rho\_base i} = \begin{pmatrix} 0 & 0 & 0 \\ \texttt{H\_aux i} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

These components directly reflect the structure included in Figure 4. Finally, the two isomorphisms included in that diagram are easily obtained from these components using the properties of the redefined reduction.

4.2.4. Formalisation of the Generalisation of the Hexagonal Lemma. The first step in the formalisation of the Generalisation of the Hexagonal Lemma is defining its hypotheses. As every module is divided into three parts, every differential consists of nine blocks.

```
Variables (m1 m2 m3: Z -> nat).
Variable Di: forall i: Z,
  'M[K]_(m1(i+1)+(m2(i+1)+m3(i+1)), (m1 i)+((m2 i)+(m3 i))).
Hypothesis boundary: forall i:Z, Di (i+1) *m Di i = 0.
Definition CH:= Build_FGChain_Complex boundary.
```

We recall that we are working with transposed matrices. Consequently, the block $(1, 2)$ of each differential, denoted by `d12`, will be an isomorphism.

```
Variable d12_1: forall i: Z, 'M[K]_(m2 i, m1 (i + 1)).
Hypothesis d12_invertible:
  forall i: Z, (d12 (i+1)) *m (d12_1 (i+1)) = 1%:M /\
               (d12_1 (i+1)) *m (d12 (i+1)) = 1%:M.
```

Afterwards, we define the morphisms `fi` and `gi` and the homotopy operator `hi` to build a reduction of the chain complex `CH`. These maps are detailed in the proof of Theorem 16. For instance, we define `gi i = ((-(d32 i)*(d12_1 i))0 1)`. Let us remark that `(gi i)` is a matrix whose type is: `'M_(m3(i+1), m1(i+1)+m2(i+1)+m3(i+1))`. In this way `(gi i)` is defined between the modules of degree `(i+1)` instead of the ones of degree `i`. This allows us to avoid casts as in the case of the definition of the differential. Then, the rest of the components of the reduction are defined accordingly. Finally, the proof of the reduction properties are obtained using essentially rewriting tactics, allowing us to build the required reduction `rhoHL`.

4.2.5. Putting together the pieces to obtain a formalisation of the BPL. In order to obtain a formalisation of the BPL, we, firstly, define the hypotheses of the lemma. Those are a reduction and a perturbation of the first chain complex of the reduction.

```
Variable K: fieldType.
Variable rho: FGReduction K.
Variable delta: forall i:Z, 'M[K]_(m (C rho)(i+1), m (C rho) i).
Hypothesis boundary_dp: forall i:Z,
  (diff (C rho) (i+1) + delta (i+1)) *m (diff (C rho) i + delta i) = 0.
```

In addition, we will assume the nilpotency hypothesis. Let us note that `pot_matrix` is a function which we have defined to compute the power of a matrix.

```
Variable (n: nat).
Hypothesis nilpotency_hp: forall i: Z,
  (pot_matrix (delta i *m Ho (H rho) i) n = 0).
```

With these hypotheses, we have to define a reduction from the chain complex with the differential of `(C rho)` perturbed by `delta`. Firstly, we apply the Decomposition Lemma over the reduction `rho`, this allows us to decompose each `(diff (C rho)i)` in the 9 blocks given in the proof of the lemma (see Subsection 4.2.3). The isomorphism given by `Fi_isom` and `Gi_isom` are used to decompose the perturbation in 9 blocks.

```
Definition deltai_new (i:Z):=
  (Fi_isom rho (i+1)) *m (delta (i+1)) *m (Gi_isom rho i).
```

Let us note that the differential `deltai_new` has moved up one degree with respect to `delta`. In this way, we obtain a division in 9 blocks of the chain complex `Di_pert i:= diff (C rho)(i+1)+ delta(i+1)`.

Then, the Generalisation of the Hexagonal Lemma can be applied if the block $(1, 2)$ of that chain complex is invertible. Following the chain of equalities included in the mathematical proof of this lemma, it is enough to prove that `1%:M + H_aux * deltai_new21` has an inverse. To this aim, the following lemma is useful – the lemma is proved using the powerful `bigop` library of SSReflect [Bertot et al. 2008].

```
Lemma inverse_I_minus_M_big (M: 'M[R]_n): (pot_matrix M m = 0) ->
  (\sum_(0<=i<m) (pot_matrix M i)) *m (1%:M - M) = 1%:M.
```

Now, applying the Generalisation of the Hexagonal Lemma `rhoHL` we build a reduction `quasi_bpl` of the decomposed and perturbed chain complex.

The Generalisation of the Hexagonal Lemma proved in Subsection 4.2.3 builds a reduction of the chain complex given as hypothesis but moved up in a degree. Moreover, the definition of `Di_pert` has required us to define it moved up in a degree more. To sum up, we have built a reduction from a perturbed chain complex but moved up two degrees `Di_BPL_up i := (C rho)(i+1+1)+ delta(i+1+1)`. Finally, in order to obtain a reduction `rho_BPL` from the initial chain complex `(C rho)` perturbed by `delta`, we can move down two degrees in the reduction obtained. For instance, the differentials are defined as `Di_BPL i:= castmx (cast3 i,cast4 i)(Di_BPL_up(i-1-1))`. In this way, we have avoided the use of casts derived from the degrees until the last step of the proof.

4.3. Using the BPL to reduce the chain complex associated with a digital image

Different ways to represent matrices exist in a system. In SSReflect, there are two available representations. The first one formalises a matrix as a function. This function determines every element of the matrix through two indices (for its row and its column). With this abstract representation it is not difficult to define different operations with matrices and prove properties of them. Indeed, an extensive library on matrices is provided in SSReflect. For this reason, this representation was chosen to prove the BPL. However, this representation is not directly executable, since this matrix definition is locked to avoid the trigger of heavy computations during deduction steps. To overcome this drawback, an alternative definition which represents matrices as lists of lists was introduced in [Dénès et al. 2012]. This concrete representation allows us to define operations which can be executed within Coq. Due to this reason, this was the representation chosen for the implementation of the RS algorithm, see Subsection 3.3. In the negative side, proving properties using this representation is much harder, since we do not have the extensive SSReflect library at our disposal.

In order to combine the best of both representations, a technique was introduced in [Dénès et al. 2012]. In that work, two morphisms are defined: `seqmx_of_mx` from abstract to concrete matrices and `mx_of_seqmx` from concrete to abstract matrices. The compositions of this morphisms are identities. In this way, it is proved that these two matrix representations are equivalent. Besides, these morphisms allow changing the representation when required. We are going to use that idea to prove the Vector Field Reduction Theorem for a chain complex generated from a digital monochromatic image. An alternative could consist in implementing an unlocked version of the SSReflect matrix as a function. In that case, a new library about matrices should be started from scratch.

The process begins by defining a new structure to represent a particular type of chain complexes, called by us *truncated chain complex*. They consist only of two matrices `d1, d2: matZ2`, whose product is null. These matrices are the only non-null components in the differential map of a chain complex. In this way, the null elements of this chain complex are not included in this definition. Besides, these matrices are represented using computable structures. The companion notions of truncated chain complex morphism, truncated homotopy operator, and truncated reduction are also defined for this structure.

As an example of the previously mentioned technique, we include the following definitions of two functions to sort a matrix, one for matrices represented as lists, and another one for matrices represented as functions (an equivalence between them is also provided in our development).

```
Definition reorderM (s1 s2: seq nat)(M: matZ2):=
 (take_columns_s s2 (take_rows_s s1 M)).


Definition reorder_mx (s1:'S_m)(s2:'S_n)(M:'M[R]_(m,n)):=
  col_perm s2 (row_perm s1 M).
```

Although these two definitions seem similar, they adopt different approaches. The first one uses simple types which are closer to a standard implementation in traditional programming languages. Indeed, that implementation is a direct translation of the one made in Haskell. We will use this version to reorder the structures when we need to compute with them. For instance, let `chaincomplexd1d2` be an initial truncated chain complex which differential is given by `d1: matZ2`, with `m` rows and `n` columns, and `d2: matZ2` with `n` rows and `p` columns. It corresponds to the representation of the initial digital image that we want to reduce. The first definition is used to obtain the ordered list of lists `d1'` and `d2'` after computing an ordered and admissible discrete vector field for `d1`.

The second definition uses the full power of dependent types and the structures and properties developed in the SSReflect library. We will use this version when we need to prove properties on the reordered structures. For instance, this definition is used to obtain the boundary property of the truncated chain complex `chaincomplexd1'd2'`, or to define an isomorphism between `chaincomplexd1d2` and `chaincomplexd1'd2'`. Both proofs take profit of properties on permutations included in the library.

We introduce just another example of the use of this approach. We need to prove that the upper-left submatrix of `d1'` is a lower triangular matrix (of dimension the number of vectors in the admissible vector field, `m1`) with 1's on its diagonal. In this case, the proof needs reasoning on the functions included in the RS algorithm and quite a long battery of ad-hoc lemmas on the computable structures are required. In a second step, we need to prove that this matrix has an inverse. This second proof is easy using the results included in the SSReflect library after changing the representation.

Now, if we want to apply the BPL, we need to build a chain complex from the ordered truncated chain complex `chaincomplexd1'd2'`. This process requires some technical steps: translate lists to SSReflect matrices, transpose the matrices, and complete with null matrices

all the not provided components. In particular, from `d'1` we define the following matrix `d'1_trmx` which type is `'M[K]_(m1+(m - m1),m1+(n - m1))`:

```
Definition d'1_trmx :=
  trmx((mx_of_seqmx (m1 + (m - m1)) (m1 + (n - m1)) d'1)).
```

Then, we build the differential of a chain complex, denoted by `CC_ordered`, in the following way:

```
Definition diff_m:= fun i:Z => match i as
 z return ((fun z0 : Z => 'M_(m_m (z0 + 1), m_m z0)) z) with
  |(-1)%Z => d0_m
  |0%Z => d'1_trmx
  |1%Z => d'2_trmx
  |2%Z => d3_m
  |3%Z => dn_m
  |_ => dn_m
end.
```

We have to highlight the differentials `d0_m` and `d3_m` because they are matrices with rows and no columns or with columns and no rows. The rest of differentials will be matrices with no rows and no columns.

We will obtain a reduction of this initial chain complex `CC_ordered` applying the BPL to the following auxiliary reduction. Let us define a chain complex `Dcc` whose modules have the same rank than the modules of `CC_ordered`, and whose differential has only one not null component. This component of type `'M[K]_(m1+(m-m1),m1+(n-m1))` is defined by a matrix `hat_d1` whose upper-left block is the identity matrix of dimension `m1` and which is null in the rest of its blocks. The reduced chain complex `Ccc` has modules whose ranks are determined by the critical cells found by the RS algorithm on `d1`. That is, the only not zero ranks are `m-m1`, `n-m1`, and `l`. The differential of this reduced chain complex is null. Then, it is easy to build a reduction between `Dcc` and `Ccc` having `h_0` (extracted from the homotopy operator given by `trmx hat_d1`) as its unique non-null component.

Now, we define a perturbation `delta_m` of `Dcc`. The non-null components of this perturbation are:

```
Definition delta_1 := d'1_trmx - hat_d1.
Definition delta_2 := d'2_trmx.
```

Then, if we prove the nilpotency condition we can apply the BPL. This lemma directly obtains the required reduction of `CC_ordered`. The natural number which allows us to prove this property is `m1.+2`. As we are working with finite structures, this number can be taken as a uniform bound for which the nilpotency condition is verified for all the elements in the module.

```
Lemma nilpotency_hp_m : forall i:Z,
  (pot_matrix(delta_m i *m (Ho (H reductionFG_gen) i))(m1.+2) = 0).
```

In the proof of this lemma, the only interesting case is when `i=0`, that is: `pot_matrix( delta_1 *m h_0)(m1.+2)= 0`. Expanding the definitions using blocks, we obtain that the only non-null blocks are `pot_matrix(d11 - id)(m1.+2)` and `d21 *m pot_matrix(d11 - id)(m1.+1)`, where `d11` and `d21` are, respectively, the up-left and down-left blocks of `d'1_trmx`.

Then, it suffices to prove that `pot_matrix(d11 - id)(m1.+1)=0`. Let us recall that `d11` is an upper triangular matrix with type `'M_m1`. We define the notion `upper_triangular_up_to_k` which given a natural number `k` and a matrix `M` determines

whether the matrix is an upper triangular matrix with 0's under and on the k-th diagonal. This definition is formalised in SSReflect as follows.

```
Definition upper_triangular_up_to_k n k (M: 'M[F]_n) :=
  forall (i j: 'I_n), j <= i + k -> M i j = 0.
```

We have developed a small library of properties on upper triangular matrices which includes the following generalisation of the lemma to prove.

```
Lemma upper_triangular_pot_matrix_n n (M: 'M[F]_n) :
  upper_triangular_up_to_k 0 M -> (pot_matrix M n.+1) = 0.
```

The BPL allows us to define a reduction `red_BPL` of `CC_ordered`. In order to obtain a truncated reduction from this reduction we retrace the technical steps made to apply the BPL: extract from this reduction the non-null maps, apply the transpose operation, and change the matrix representation. For instance, the two components of the first truncated chain complex in the reduction are defined in the following way.

```
Definition d'1_trmx_trmx:= seqmx_of_mx (trmx (diff (C red_BPL) 0)).
Definition d'2_trmx_trmx:= seqmx_of_mx (trmx (diff (C red_BPL) 1)).
```

It is necessary to stress that the obtained big truncated chain complex is composed of the differentials where the transpose operation has been applied twice. Some casts are needed in a last step in order to build a truncated reduction of the chain complex `chaincomplexd 'd'2`.

Finally, to obtain the required reduction, it remains to compose the isomorphism between the truncated chain complex `chaincomplexd1d2` obtained from a digital image and the ordered truncated chain complex `chaincomplexd1'd2'` with the reduction built as explained in the last paragraph.

5. CERTIFIED COMPUTATION OF THE HOMOLOGY ASSOCIATED WITH A DIGITAL IMAGE

From a theoretical point of view, our objective consists in developing a formal proof which verifies that the homology of a chain complex produced by actual images as the one of Figure 2 and its reduced chain complex have the same homology; this has been accomplished in the previous sections. But, as we are working in a constructive setting, we could also undertake the computation inside Coq of the reduced chain complex. In the development included in the previous sections, we have developed a proof of the BPL for general (finite type) chain complexes, providing a complete formalisation. The reduced matrix is then obtained through the BPL, which does not use executable matrices. We recall that we have worked with the matrix definition included in the SSReflect library in order to prove the BPL and that this definition is locked to avoid the trigger of heavy computations during deduction steps. It makes this general method not directly executable.

We can improve the previous approach, when particularising it to a concrete situation: we consider truncated chain complexes obtained from digital images, and we obtain an admissible discrete vector field using the RS algorithm on one of the matrices $d1$ of the differential map. In this situation, it is possible to obtain a reduction of $d1$ using directly the Hexagonal Lemma [Romero and Sergeraert 2010] (instead of the BPL). Concretely, the initial matrix $d1$ is divided into 4 blocks and the reduced one is built by the formula $d22 - d21 * d11^{-1} * d12$. Finally, the reduction is extended to the other non-null matrix of the differential in order to preserve the boundary condition. In order to obtain an executable version of that reduction we have also formalised this simplified proof using computable structures. With this version we are able to compute the whole process for toy examples.

When the size of the matrix grows, it is not possible to compute the full path directly inside Coq. But we can calculate, in a certified manner within Coq, the homology of the

reduced matrix. That is the case of actual images as the one in Figure 2. In this case, the original matrix has dimensions 690×1400 and the reduced one 97×500. This reduced matrix has been obtained using the Haskell version of the algorithm formalised in Coq. With the current state of the running machinery in Coq, the system is not able to compute of the steps producing this reduced chain complex. The bottleneck of our development is the definition of the inverse of a matrix. Although we have used a specialized function to compute the inverse of a triangular matrix with 1's on its diagonal, implemented in [Dénès et al. 2013], the performance gain has not been enough in this particular application. (Let us observe that this bottleneck occurs due to the difficulties of handling big data structures in proofs assistants, and not due to theoretical complexity issues.) Finally, the solution adopted was to use Haskell as an oracle. The process is the following. From the matrices of the chain complex of a digital image, we delegate in Haskell the computation of the reduced matrix and the matrices of the required reduction (they also include the inverse matrix in their definition). Then, these matrices are brought into Coq which obtains the homology (in 25 seconds) of the reduced matrix and automatically compute (using the `vm_compute` tactic) the proofs (in 8 hours) of the reduction between both matrices. These running times are clearly unsatisfactory (in our software in production, response times of seconds are required). Thus, more research efforts are needed as we explain in the next section.

6. CONCLUSIONS AND FURTHER WORK

In this paper, we have reported on a research providing a certified computation of homology groups associated with some digital images coming from a biomedical problem. The whole Coq development contains approximately 45000 lines of code, 2500 theorems, and 1700 definitions. The main contributions allowing us to reach this challenging goal have been:

- The implementation in Coq of Romero-Sergeraert's algorithm [Romero and Sergeraert 2010] computing a discrete vector field for a digital image.
- The complete formalisation in Coq of a proof of the Basic Perturbation Lemma (to be compared with the one at [Aransay et al. 2008]).
- A methodology to overcome the problems of efficiency related to the execution of programs inside proof assistants; in this approach the programming language Haskell has been used with two different purposes: as a fast prototyping tool and as an oracle.

Emphasis has been put in the execution of proofs inside Coq, to show explicitly the computability of our approach, but without the aim of using any of these proofs as internal decision procedures. Since the final execution on real examples requires using Haskell oracles, the question of why verifying the whole process (and not only the oracle programs) arises. Our point of view is that both aspects are approaching different concerns about increasing reliability: the verification of the complete path is related to the correctness of the *algorithm*, while the oracle is ensuring correctness of the application on a series of *concrete* instances (images), in a kind of *formal testing*. Thus, both contributions are relevant and complementary.

As for future work, several problems remain open. The most evident one is obtaining a better performance in the process. This can be undertaken at three different levels. First, by using other algorithms to compute the main objects in our approach (discrete vector fields of digital images, inverses of matrices, and so on). Second, by implementing more efficient data structures, suitable for the theorem proving setting. For instance, we can explore recent work on (experimental) native arrays [Boespflug et al. 2011] in Coq. Third, by improving the running environments in proof assistants.

Another option consists in using the Coq extracting code mechanism [Letouzey 2008]. Nowadays, the functional languages available as output include Haskell. This line has not been explored in this paper and it is proposed as future work. For instance, a performance

comparison among the extracted code, the hand-written code and the direct execution inside Coq could deserve attention.

Another line of research is to apply our methodology and techniques to other problems related to the homological processing of biomedical images. The best candidate is *persistent homology*, which has been already applied and formalised (see [Heras et al. 2013]). The project would be to study whether our reduction strategy can be also profitable in this new homological context.

REFERENCES

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

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

M. Armand, B. Grégoire, A. Spiwack, and L. Théry. 2010. Extending Coq with imperative features and its application to SAT verification. In *Proceedings 1st International Conference on Interactive Theorem Proving (ITP'10) (Lecture Notes in Computer Science)*, Vol. 6172. 83–98.

R. Ayala, E. Domínguez, A.R. Francés, and A. Quintero. 2003. Homotopy in digital spaces. *Discrete Applied Mathematics* 125 (2003), 3–24.

D. Barnes and L. Lambe. 1991. Fixed point approach to homological perturbation theory. *Proc. Amer. Math. Soc.* 112, 3 (1991), 881–892.

G. Barthe and P. Courtieu. 2002. Efficient Reasoning about Executable Specifications in Coq. In *Proceedings 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLS'02) (Lecture Notes in Computer Science)*, Vol. 2410. 31–46.

M. Bear, B. Connors, and M. Paradiso. 2006. *Neuroscience: Exploring the Brain*. Lippincott Williams & Wilkins.

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

Y. Bertot, G. Gonthier, S. Ould Biha, and I. Pasca. 2008. Canonical Big Operators. In *Proceedings 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLS'08) (Lecture Notes in Computer Science)*, Vol. 5170. 86–101.

M. Boespflug, M. Dénès, and Grégoire. 2011. Full Reduction at Full Throttle. In *Proceedings Certified Programs and Proofs (Lecture Notes in Computer Science)*, Vol. 7086. 362–377.

F. Cazals, F. Chazal, and T. Lewiner. 2003. Molecular shape analysis based upon Morse-Smale complex and the Connolly function. In *Proceedings 19th ACM Symposium on Computational Geometry (SCG'03)*. 351–360.

K. Claessen and J. Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings 5th ACM SIGPLAN international conference on Functional programming (ICFP'00)*. ACM Press, 268–279.

G. Cuesto and others. 2011. Phosphoinositide-3-Kinase Activation Controls Synaptogenesis and Spinogenesis in Hippocampal Neurons. *The Journal of Neuroscience* 31, 8 (2011), 2721–2733.

M. Dénès, A. Mörtberg, and V. Siles. 2012. A Refinement Based Approach to Computational Algebra in Coq. In *Proceedings 3rd International Conference on Interactive Theorem Proving (ITP'2012) (Lecture Notes in Computer Science)*, Vol. 7406. 83–98.

M. Dénès, A. Mörtberg, and V. Siles. 2013. CoqEAL, the Coq Effective Algebra Library. (2013). http://www.maximedenes.fr/content/coqeal-coq-effective-algebra-library.

C. Domínguez and J. Rubio. 2011. Effective Homology of Bicomplexes, formalized in Coq. *Theoretical Computer Science* 412 (2011), 962–970.

X. Dousson, J. Rubio, F. Sergeraert, and Y. Siret. 1998. The Kenzo program. Institut Fourier, Grenoble. http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/.

H. Edelsbrunner and J. L. Harer. 2010. *Computational Topology: An Introduction*. American Mathematical Society.

R. Forman. 1998. Morse theory for cell complexes. *Advances in Mathematics* 134 (1998), 90–145.

G. Gonthier. 2008. Formal proof - The Four-Color Theorem. *Notices of the American Mathematical Society* 55, 11 (2008), 1382–1393.

G. Gonthier and A. Mahboubi. 2010. An introduction to small scale reflection in Coq. *Journal of Formal Reasoning* 3, 2 (2010), 95–152.

B. Grégoire and X. Leroy. 2002. A compiled implementation of strong reduction. In *Proceedings International Conference on Functional Programming 2002*. ACM Press, 235–246.

A. Gyulassy, P.T. Bremer, B. Hamann, and V. Pascucci. 2008. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1619–1626.

J. Heras and others. 2012a. Towards a certified computation of homology groups for digital images. In *Proceedings 4th International Workshoph on Computational Topology in Image Context (CTIC'12) (Lecture Notes in Computer Science)*, Vol. 7309. 49–57.

J. Heras and others. 2012b. Verifying an algorithm computing Discrete Vector Fields for digital imaging. In *Proceedings 19th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning (Calculemus'12) (Lecture Notes in Computer Science)*, Vol. 7362. 215–229.

J. Heras, T. Coquand, A. Mörtberg, and V. Siles. 2013. Computing Persistent Homology within Coq/SSReflect. *ACM Transactions on Computational Logic* 14, 4 (2013), 26.

J. Heras, G. Mata, M. Poza, and J. Rubio. 2011. Homological processing of biomedical digital images: automation and certification. *Imagen-A* 2, 4 (2011), 29–31.

G. Hutton. 2007. *Programming in Haskell*. Cambridge University Press.

N. Jacobson. 1989. *Basic Algebra II*. W. H. Freeman and Company.

G. Jerse and N. M. Kosta. 2010. Tracking features in image sequences using discrete Morse functions. In *Proceedings 3rd International Workshop on Computational Topology in Image Context (CTIC'10) (Imagen-A)*, Vol. 1. 27–32.

D. Kozlov. 2007. *Combinatorial Algebraic Topology*. Springer.

R. Krebbers and B. Spitters. 2011. Computer Certified Efficient Exact Reals in Coq. In *Proceedings Conferences Intelligent Computer Mathematics (CICM'11) (Lecture Notes in Computer Science)*, Vol. 6824. 90–106.

P. Letouzey. 2008. Coq Extraction, an Overview. In *Proceedings Logic and Theory of Algorithms, Fourth Conference on Computability in Europe (CiE'08) (Lecture Notes in Computer Science)*, Vol. 5028. 359–369.

T. Lewiner, H. Lopes, and G. Tavares. 2004. Applications of Forman's discrete morse theory to topology visualization and mesh compression. *Transactions on Visualization and Computer Graphics* 10, 5 (2004), 499–508.

A. Mörtberg. 2010. Constructive algebra in functional programming and type theory. In *Mathematics, Algorithms and Proofs 2010*.

A. Romero and F. Sergeraert. 2010. Discrete Vector Fields and Fundamental Algebraic Topology. http://arxiv.org/abs/1005.5685v1.

A. Romero and F. Sergeraert. 2012. Homological Perturbation Theorem and Eilenberg-Zilber vector field. http://www-fourier.ujf-grenoble.fr/~sergerar/Talks/12-06-Zurich-1.pdf.

J. Rubio and F. Sergeraert. 1997. Constructive Algebraic Topology, Lecture Notes Summer School on Fundamental Algebraic Topology. Institut Fourier, Grenoble. http://www-fourier.ujf-grenoble.fr/~sergerar/Summer-School/.

J. Rubio and F. Sergeraert. 2002. Constructive Algebraic Topology. *Bulletin des Sciences Mathématiques* 126, 5 (2002), 389–412.

D. J. Selkoe. 2002. Alzheimer's disease is a synaptic failure. *Science* 298, 5594 (2002), 789–791.

D. Ziou and M. Allili. 2002. Generating Cubical Complexes from Image Data and Computation of the Euler number. *Pattern Recognition* 35 (2002), 2833–2839. Issue 12.