Verifying an algorithm computing Discrete Vector Fields for digital imaging*

Jónathan Heras, María Poza, and Julio Rubio

Department of Mathematics and Computer Science of University of La Rioja
{jonathan.heras, maria.poza, julio.rubio}@unirioja.es

Abstract. In this paper, we present a formalization of an algorithm to construct *admissible discrete vector fields* in the Coq theorem prover taking advantage of the SSReflect library. Discrete vector fields are a tool which has been welcomed in the *homological analysis* of digital images since it provides a procedure to reduce the amount of information but preserving the homological properties. In particular, thanks to discrete vector fields, we are able to compute, inside Coq, homological properties of *biomedical images* which otherwise are out of the reach of this system.

Keywords: Discrete Vector Fields, Haskell, Coq, SSReflect, Integration

1 Introduction

Kenzo [10] is a Computer Algebra System devoted to Algebraic Topology which was developed by F. Sergeraert. This system has computed some homology and homotopy groups which cannot be easily obtained by theoretical or computational means; some examples can be seen in [23]. Therefore, in this situation, it makes sense to analyze the Kenzo programs in order to ensure the correctness of the mathematical results which are obtained thanks to it. To this aim, two different research lines were launched some years ago to apply formal methods in the study of Kenzo.

On the one hand, the ACL2 theorem prover has been used to verify the correctness of actual Kenzo *programs*, see [17,21]. ACL2 fits perfectly to this task since Kenzo is implemented in Common Lisp [14], the same language in which ACL2 is built on. Nevertheless, since the ACL2 logic is first-order, the full verification of Kenzo is not possible, because it uses intensively higher order functional programming. On the other hand, some instrumental Kenzo *algorithms*, involving higher-order logic, have been formalized in the proof assistants Isabelle/HOL and Coq. Namely, we can highlight the formalizations of the Basic Perturbation Lemma in Isabelle/HOL, see [2], and the Effective Homology of Bicomplexes in Coq, published in [9].

* Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by the European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

The work presented in this paper goes in the same direction that the latter approach, formalizing Kenzo *algorithms*. In particular, we have focused on the formalization of *Discrete Vector Fields*, a powerful notion which will play a key role in the new version of Kenzo; see the Kenzo web page [10]. To carry out this task, we will use the Coq proof assistant [7] and its SSReflect library [13].

The importance of Discrete Vector Fields, which were first introduced in [11], stems from the fact that they can be used to considerably reduce the amount of information of a discrete object but preserving homological properties. In particular, we can use discrete vector fields to deal with biomedical images inside Coq in a reasonable amount of time.

The rest of this paper is organized as follows. In the next section, we introduce some mathematical preliminaries, which are encoded abstractly in Coq in Section 3. Such an abstract version is *refined* to an effective one in Section 4; namely, the implementation and formal verification of the main algorithm involved in our developments are presented there. In order to ensure the feasibility of our programs, a major issue when applying formal methods, we use them to study a biomedical problem in Section 5. The paper ends with a section of Conclusions and Further work, and the Bibliography.

The interested reader can consult the complete development in `http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/ProofExamples#wp3ex5`.

2 Mathematics to formalize

In this section, we briefly provide the minimal mathematical background needed to understand the rest of the paper. We mainly focus on definitions which, mainly, come from the algebraic setting of discrete Morse theory presented in [24] and the Effective Homology theory [25]. We assume as known the notions of *ring*, *module* over a ring and *module morphism* (see, for instance, [18]).

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 of sequences $(C_n, d_n)_{n\in\mathbb{Z}}$ where for every $n \in \mathbb{Z}$, C_n is an $\mathcal{R}$-module and $d_n : C_n \to C_{n-1}$ is a module morphism, called the *differential map*, such that the composition $d_n d_{n+1}$ is null. In many situations the ring $\mathcal{R}$ is either the integer ring, $\mathcal{R} = \mathbb{Z}$, or the field $\mathbb{Z}_2$. In the rest of this section, we will work with $\mathbb{Z}$ as ground ring; later on, we will change to $\mathbb{Z}_2$.
The module C_n is called the module of *n-chains*. 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 for every $n \in \mathbb{Z}$ the relation $f_{n-1}d_n = \widehat{d}_n f_n$. Usually, the sub-indexes are skipped, and we just write $fd_C = d_D f$.

Now, we can introduce one of the fundamental notions in the effective homology theory.

Definition 4 A *reduction* ρ between two chain complexes C_* and D_*, denoted in this paper 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 relations are satisfied:

1) $f \circ g = Id_{D_*}$;
2) $d_C \circ h + h \circ d_C = Id_{C_*} - g \circ f$;
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_*.

Let us state now the main notions coming from the algebraic setting of Discrete Morse Theory [24].

Definition 5 Let $C_* = (C_n, d_n)_{n\in\mathbb{Z}}$ be a free chain complex with distinguished $\mathbb{Z}$-basis $\beta_n \subset C_n$. A *discrete vector field* V on C_* is a collection of pairs $V = \{(\sigma_i; \tau_i)\}_{i\in I}$ 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 generator (*cell*) of C_* appears at most one time in V.

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

Definition 6 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*.

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

Definition 7 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 a 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 $\sigma_{i_{k-1}}$.

Now we can present the notion of *admissible* discrete vector field on a chain complex, a concept which can be understood as a *recipe* indicating both the "useless" elements of the chain complex (in the sense, that they can be removed without changing its homology) and the *critical* ones (those whose removal modifies the homology).

Definition 8 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 $\lambda_n(\sigma)$.

Finally, we can state the theorem where Discrete Morse Theory and Effective Homology converge.

Theorem 9 [24, Theorem 19] 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 $\rho : (C_n, d_n) \Rightarrow (C^c_n, d^c_n)$ where $C^c_n = \mathbb{Z}[\beta^c_n]$ is the free $\mathbb{Z}$-module generated by β^c_n, the critical n-cells.

Therefore, as the bigger the admissible discrete vector field V the smaller the chain complex C^c_*, we need algorithms which produce admissible discrete vector fields as large as possible.

If we consider the case of *finite type* chain complexes, where there is a finite number of generators in each dimension of the chain complex, the differential maps can be represented as matrices. In that case, the problem of finding an admissible discrete vector field on the chain complex can be solved through the computation of an admissible vector field for those matrices.

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 this matrix is a set of natural pairs $\{(a_i, b_i)\}$ satisfying these conditions:

1. $1 \leq a_i \leq m$ and $1 \leq b_i \leq n$.
2. The entry $M[a_i, b_i]$ of the matrix is ± 1.
3. The indexes a_i (resp. b_i) are pairwise different.

Given V be a vector field for our matrix M, we need to know if V is admissible. If $1 \leq a, a' \leq m$, with $a \neq a'$, we can decide $a > a'$ if there is an elementary V-path from a to a', that is, if a vector (a, b) is present in V and the entry $M[a', b]$ is non-null. In this way, a binary relation is obtained. Then the vector field V is admissible if and only if this binary relation generates a partial order, that is, if there is no loop $a_1 > a_2 > \ldots > a_k = a_1$.

Eventually, given a matrix and an admissible discrete vector field on it, we can construct a new matrix, smaller than the original one, preserving the homological

properties. This is the equivalent version of Theorem 9 for matrices, a detailed description of the process can be seen in [24, Proposition 14].

In the rest of the paper, we will focus on the formally certified construction of an admissible discrete vector field from a matrix. The task of verifying the reduction process remains as further work.

3 A non deterministic algorithm in SSReflect

First of all, we have provided in Coq/SSReflect an abstract formalization of admissible discrete vector fields on matrices and a non deterministic algorithm to construct an admissible discrete vector field from a matrix[1]. SSReflect is an extension for the Coq proof assistant, which was developed by G. Gonthier while formalizing the Four Color Theorem [12]. Nowadays, it is used in the formal proof of the Feit-Thompson theorem [1].

SSReflect provides all the necessary tools to achieve our goal. In particular, we take advantage of the `matrix`, `ssralg` and `fingraph` libraries, which formalize, respectively, matrix theory, the main algebraic structures and the theory of finite graphs.

First of all, we are going to define admissible discrete vector field on a matrix M with coefficients in a ring $\mathcal{R}$, and with m rows and n columns. It is worth noting that our matrices are defined over a generic ring instead of working with coefficients in $\mathbb{Z}$ since the SSReflect implementation of $\mathbb{Z}$, see [6], is not yet included in the SSReflect distributed version. The vector fields are represented by a sequence of pairs where the first component is an ordinal m and the second one an ordinal n.

```
Variable R : ringType.
Variables m n : nat.
Definition vectorfield := seq ('I_m * 'I_n).
```

Now, we can define in a straightforward manner a function, called `dvf`, which given a matrix `M` (with coefficients in a ring $\mathcal{R}$, and with m rows and n columns, `'M[R]_(m,n)`) and an object `V` of type `vectorfield` checks whether `V` satisfies the properties of a discrete vector field on `M` (Definition 10).

```
Definition dvf (M : 'M[R]_(m,n)) (V : vectorfield) :=
  all [pred p | (M p.1 p.2 == 1) || (M p.1 p.2 == -1)] V &&
  (uniq (map (@fst _ _) V) && uniq (map (@snd _ _) V)).
```

It is worth noting that the first condition of Definition 10 is implicit in the `vectorfield` type. Now, as we have explained at the end of the previous section, from a discrete vector field `V` a binary relation is obtained between the first elements of each pair of `V`. Such a binary relation will be encoded by means of an object of the following type.

[1] Thanks are due to Maxime Dénès and Anders Mörtberg which guided us in this development.

```
Definition orders := (simpl_rel 'I_m).
```

Finally, we can define a function, which is called `advf`, that given a matrix `'M[R]_(m,n)`, `M`, a `vectorfield`, `V` and an `orders`, `ords`, as input, tests whether both `V` satisfies the properties of a discrete vector field on `M` and the admissibility property for the relations, `ords`, associated with the vector field, `V`. In order to test the admissibility property we generate the transitive closure of `ords`, using the `connect` operator of the `fingraph` library, and subsequently check that there is not any path between the first element of a pair of `V` and itself.

```
Definition advf (M:'M[R]_(m,n)) (V:vectorfield) (ords:orders) :=
 dvf M V && all [pred i|~~(connect ords i i)] (map (@fst _ _) V).
```

Now, let us define a non deterministic algorithm which construct an admissible discrete vector field from a matrix. Firstly, we define a function, `gen_orders`, which generates the relations between the elements of the discrete vector field as we have explained at the end of the previous section.

```
Definition gen_orders (M0 : 'M[R]_(m,n)) (i:'I_m) j :=
  [rel i x | (x != i) && (M0 x j != 0)].
```

Subsequently, the function, `gen_adm_dvf`, which generates an admissible discrete vector field from a matrix is introduced. This function invokes a recursive function, `genDvfOrders`, which in each step adds a new component to the vector field in such a way that the admissibility property is fulfilled. The recursive algorithm stops when either there is not any new element whose inclusion in the vector field preserves the admissibility property or the maximum number of elements of the discrete vector field (which is the minimum between the number of columns and the number of rows of the matrix) is reached.

```
Fixpoint genDvfOrders M V (ords : simpl_rel _) k :=
  if k is l.+1 then
    let P := [pred ij | admissible (ij::V) M
                       (relU ords (gen_orders M ij.1 ij.2))] in
    if pick P is Some (i,j)
       then genDvfOrders M ((i,j)::V)
                         (relU ords (gen_orders M i j)) l
    else (V, ords)
  else (V, ords).

Definition gen_adm_dvf M :=
  genDvfOrders M [::] [rel x y | false] (minn m n).
```

Eventually, we can certify in a straightforward manner (just 4 lines) the correctness of the function `gen_adm_dvf`.

```
Lemma admissible_gen_adm_dvf m n (M : 'M[R]_(m,n)) :
 let (vf,ords) := gen_adm_dvf M in admissible vf M ords.
```

As a final remark, it is worth noting that the function `gen_adm_dvf` is not executable. On the one hand, SSReflect matrices are locked in a way that

do not allow direct computations since they may trigger heavy computations during deduction steps. On the other hand, we are using the `pick` instruction, in the definition of `genDvfOrders`, to choose the elements which are added to the vector field; however, this operator does not provide an actual method to select those elements.

4 An effective implementation: from Haskell to Coq

In the previous section, we have presented a non deterministic algorithm to construct an admissible discrete vector field from a matrix. Such an abstract version has been described on high-level datastructures; now, we are going to obtain from it a *refined* version, based on datastructures closer to machine representation, which will be executable.

The necessity of an executable algorithm which construct an admissible discrete vector field stems from the fact that they will will play a key role to study biomedical images. There are several algorithms to construct an admissible discrete vector field; the one that we will use is explained in [24] (from now on, called RS's algorithm; RS stands for Romero-Sergeraert). The implementation of this algorithm will be executable but the proof of its correctness will be much more difficult than the one presented in the previous section.

4.1 The Romero-Sergeraert algorithm

The underlying idea of the RS algorithm is that given an admissible discrete vector field, we try to enlarge it adding new vectors which preserve the admissibility property. We can define algorithmically 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 for M.
Description:

1. Initialize the vector field, V, to the void vector field.
2. Initialize the relations, *ords*, to nil.
3. For every row, i, of M:
   3.1. Search the first entry of the row equal to 1 or -1, j.
   3.2. `If` (i, j) can be added to the vector field; that is, if we add it to V and generate all the relations, the properties of an admissible discrete vector field are preserved.
      `then`:
        - Add (i, j) to V.
        - Add to *ords* the corresponding relations generated from (i, j).
        - Go to the next row and repeat from Step 3.
      `else`: look for the next entry of the row whose value is 1 or -1.
        - `If` there is not any.

`then`: go to the next row and repeat from Step 3.
`else`: go to Step 3.2 with j the column of the entry whose value is 1 or -1.

In order to clarify how this algorithm works, let us construct an admissible discrete vector field from the following matrix.

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

We start with the void vector field $V = \{\}$. Running the successive rows, we find $M[1,1] = 1$, and we include the vector $(1,1)$ to V, obtaining $V = \{(1,1)\}$. Then, let us add the relations that, in this case is $1 > 2$ because $M[2,1] \neq 0$. So, it will be forbidden to incorporate the relation $2 > 1$ as the cycle $1 > 2 > 1$ would appear. Besides, the row 1 and the column 1 are now used and cannot be used anymore. So, we go on with the second row and find $M[2,1] = 1$, but we cannot add $(2,1)$ as we have just said. Moreover, the element $(2,2)$ can not be incorporated because the cycle $1 > 2 > 1$ would be created. So, we continue and find the next element, $M[2,3] = 1$. This does not create any cycle and satisfies the properties of a discrete vector field. Then, we obtain $V = \{(2,3),(1,1)\}$ and the relations $2 > 3$ and $2 > 4$. Running the next row, the first element equal to 1 is in the position $(3,3)$, but we cannot include it due to the admissibility property. Therefore, we try with the last element of this row $(3,4)$. No relation is generated in this case because in this column the only non null element is in the chosen position. So, $V = \{(3,4),(2,3),(1,1)\}$. Finally, we run the last row. The elements that could be added are $(4,2),(4,3)$, but in both cases we would have to append the relation $4 > 2$. This would generate a cycle with one of the previous restrictions, $2 > 4$. So, we obtain $V = \{(3,4),(2,3),(1,1)\}$ and the relations are: $1 > 2$, $2 > 3$ and $2 > 4$.

In general, Algorithm 11 can be applied over matrices with coefficients in a general ring $\mathcal{R}$. From now on, we will work with $\mathcal{R} = \mathbb{Z}_2$, since this is the usual ring when working with monochromatic images in the context of Digital Algebraic Topology.

The development of a formally certified implementation of the RS algorithm has followed the methodology presented in [22]. Firstly, we implement a version of our programs in *Haskell* [19], a *lazy* functional programming language. Subsequently, we intensively test our implementation using *QuickCheck* [5], a tool which allows one to automatically test properties about programs implemented in Haskell. Finally, we verify the correctness of our programs using the Coq *interactive* proof assistant and its SSReflect library.

4.2 A Haskell program

The choice of Haskell to implement our programs was because both the code and the programming style is similar to the ones of Coq . In this programming

language, we have defined the programs which implement the RS algorithm. The description of the main function is shown here:

gen_admdvf_ord M**:** From a matrix M with coefficients in $\mathbb{Z}_2$, represented as a list of lists, this function generates an admissible discrete vector field for M, encoded by a list of natural pairs, and the relations, a list of lists of natural numbers.

Let us emphasize that the function `gen_admdvf_ord` returns a pair of elements. The former one, `(gen_admdvf_ord M).1`, is a discrete vector field and the latter one, `(gen_admdvf_ord M).2`, corresponds to the relations associated with the vector field. To provide a better understanding of these tools, let us apply them in the example presented in Subsection 4.1.

```
> gen_admdvf_ord [[1,1,0,0],[1,1,1,0],[0,0,1,1],[0,1,1,0]]
[([(3,4),(2,3),(1,1)], [[2,4],[2,3],[1,2],[1,2,4],[1,2,3]])]
```

Let us note that we return the transitive closure of the relations between the first components of the pairs of the discrete vector field. This will make the proof of the correctness of our programs easier.

4.3 Testing with QuickCheck

Using QuickCheck can be considered as a good starting point towards the formal verification of our programs. On the one hand, a specification of the properties which must be satisfied by our programs is given (a necessary step in the formalization process). On the other hand, before trying a formal verification of our programs (a quite difficult task) we are testing them, a process which can be useful in order to detect bugs.

In our case, we want to check that the output by `gen_admdvf_ord` gives us an admissible discrete vector field. Then, let M be a matrix over $\mathbb{Z}_2$ with m rows and n columns, $V = (a_i, b_i)_i$ be a discrete vector field from M and $ords$ be the transitive closure of the relations associated with V, the properties to test are the ones coming from Definition 10 and the admissibility property adapted to the $\mathbb{Z}_2$ case.

1. $1 \leq a_i \leq m$ and $1 \leq b_i \leq n$.
2. $\forall i$, $M(a_i, b_i) = 1$.
3. $(a_i)_i$ (resp. $(b_i)_i$) are pairwise different.
4. $ords$ does not have any loop (admissibility property).

These four properties has been encoded in Haskell by means of a function called `isAdmVecfield`. To test in QuickCheck that our implementation of the RS algorithm fulfills the specification given in `isAdmVecfield`, the following *property definition*, using QuickCheck terminology, is defined.

```
condAdmVecfield M =
let advf = (gen_admdvf_ord M) in isAdmVecfield M (advf.1) (advf.2)
```

The definition of `condAdmVecfield` states that given a matrix `M`, the output of `gen_admdvf_ord`, both the discrete vector field (first component) and the relations (second component) from M, fulfill the specification of the property called `isAdmVecfield`. Now, we can test whether `condAdmVecfield` satisfies such a property.

```
> quickCheck condAdmVecfield
+ + + OK, passed 100 tests.
```

The result produced by QuickCheck when evaluating this statement, means that QuickCheck has generated 100 random values for `M`, checking that the property was true for all these cases. To be more precise, QuickCheck generates 100 random matrices over $\mathbb{Z}$; however, our algorithm works with matrices over $\mathbb{Z}_2$, then to overcome this pitfall we have defined a function which transforms matrices over $\mathbb{Z}$ into matrices over $\mathbb{Z}_2$ turning even entries of the matrices into 0's and odd entries into 1's. In this way, we obtain random matrices over $\mathbb{Z}_2$ to test our programs.

4.4 Formalization in Coq /SSReflect

After testing our programs, and as a final step to confirm their correctness, we can undertake the challenge of formally verifying them.

First of all, we define the data types related to our programs, which are effective matrices, vector fields and relations. We have tried to keep them as close as possible to the Haskell ones; therefore, a matrix is represented by means of a list of lists over $\mathbb{Z}_2$, a vector field is a sequence of natural pairs and finally, the relations is a list of lists of natural numbers.

```
Definition Z2 := Fp_fieldType 2.
Definition matZ2 := seq (seq Z2).
Definition vectorfield := seq (prod nat nat).
Definition orders := seq (seq nat).
```

Afterwards, we translate both the programs and the properties, which were specified during the testing of the programs, from Haskell to Coq, a task which is quite direct since these two systems are close.

Then, we have defined a function `isAdmVecfield` which receives as input a matrix over $\mathbb{Z}_2$, a vector field and the relations and checks if the properties, explained in Subsection 4.3, are satisfied.

```
Definition isAdmVecfield (M:matZ2)(vf:vectorfield)(ord:orders):=
  ((longmn (size M) (getfirstElemseq vf)) /\
  (longmn (size (nth [::] M 0)) (getsndElemseq vf))) /\
  (forall i j:nat, (i , j) \in vf -> compij i j M = 1) /\
  ((uniq (getfirstElemseq vf)) /\ (uniq (getsndElemseq vf))) /\
  (admissible ord).
```

Finally, we have proved the theorem `genDvfisVecfield` which says that given a matrix M, the output produced by `gen_admdvf_ord` satisfies the properties specified in `isAdmVecfield`.

```
Theorem genDvfisVecfield (M:matZ2):
  let advf := (gen_admdvf_ord M) in
  isAdmVecfield M (advf.1) (advf.2).
```

We have split the proof of the above theorem into 4 lemmas which correspond with 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):
  let advf := (gen_admdvf_ord M).1 in
  (longmn (size M) (getfirstElemseq advf) /\
  (longmn (size (nth nil M 0))(getsndElemseq advf)).
```

Both the functions which implement the RS algorithm and the ones which specify the definitional 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 [3] which allows one to reason about complex recursive definitions since Coq does not directly generate elimination principles for complex recursive functions. Let us see how the tool presented in [3] works.

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.
```

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 induction 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 to prove this theorem.

```
Fixpoint Mij_subM_rec (i k: nat) (M: matZ2) :=
match k with
```

```
|O => 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.
```

This style of proving functional programs in Coq is the one followed in the development of the proof of Theorem `genDvfisVecfield`.

4.5 Experimental results

Using the same methodology presented throughout this section, we are working in the formalization of the algorithm which, from a matrix and an admissible discrete vector field on it, produces a reduced matrix preserving the homological properties of the original one. Up to now, we have achieved a Haskell implementation which has been both tested with QuickCheck and translated into Coq; however, the proof of its correctness remains as further work.

Anyway, as we have a Coq implementation of that procedure, we can execute some examples inside this proof assistant. Namely, we have integrated the programs presented in this paper with the ones devoted to the computation of homology groups of digital images introduced in [15]. The reason to carry out the computations within Coq, instead of transferring the verified algorithms into a more efficient programming language like OCaml or Haskell, is the chance that this approach provides us to reuse the result of our homological computations for further proofs. It is worth noting that the outputs produced by external programs are untrusted so they cannot be imported inside Coq.

As a benchmark we have considered matrices coming from 500 randomly generated images. The size of the matrices associated with those images was initially around 100×300, and after the reduction process the average size was 5×50. Using the original matrices Coq takes around 12 seconds to compute the homology from the matrices; on the contrary, using the reduced matrices Coq only needs milliseconds. Furthermore, as we will see in the following section, we have studied some images which are originated from a real biomedical problem.

5 Application to biomedical images

Biomedical images are a suitable benchmark for testing our programs. On the one hand, 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. On the other hand, 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 [4] are the points of connection between neurons and are related to the computational capabilities of the brain. Therefore, the treatment of neurological diseases, such as Alzheimer, may take advantage of procedures modifying the number of synapses [8].

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 [16]. 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), a monochromatic image is obtained, see Figure 1[2]. In such an image, each connected component represents a synapse. So, the problem of measuring the number of synapses is translated into a question of counting the connected components of a monochromatic image.

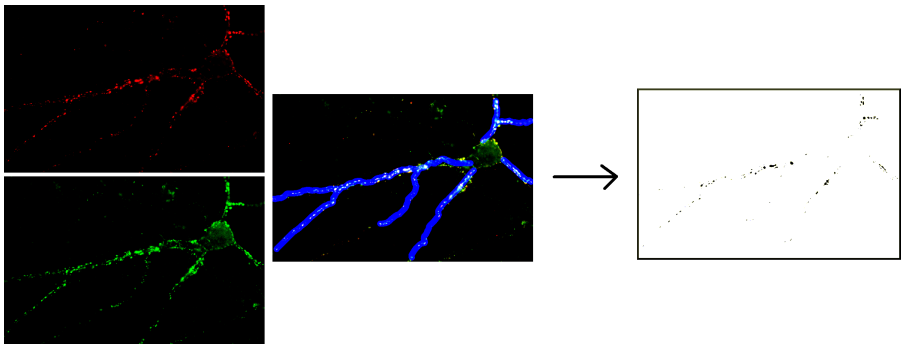

Fig. 1. Synapses extraction from three images of a neuron

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 which were presented in [15]. Nevertheless, such programs are not able to handle images like the one of the right side of Figure 1 due to its size (let us remark that Coq is a proof assistant tool and not a computer algebra system). In order to overcome this drawback, as we have explained at the end of the previous section, we have integrate our reduction programs with the ones presented in [15]. Using this approach, we can successfully compute the homology of the biomedical images in just 25 seconds, a remarkable time for an execution inside Coq.

[2] The same images with higher resolution can be seen in `http://www.unirioja.es/cu/joheras/synapses/`

6 Conclusions and Further work

In this paper, we have given the first step towards the formal verification of a procedure which allows one to study homological properties of big digital images inside Coq. The underlying idea consists in building an admissible discrete vector field on the matrices associated with an image and, subsequently reduce those matrices but preserving the homology.

Up to now, we have certified the former step of this procedure, the construction of an admissible discrete vector field from a matrix, both in an abstract and a concrete way. The reason because the abstract formalization is useful is twofold: on the one hand, it provides a high-level theory close to usual mathematics, and, on the other hand, it has been refined to obtain the effective construction of admissible discrete vector fields. As we have explained, there are several heuristics to construct an admissible discrete vector field from a matrix, the one that we have chosen is the RS algorithm [24] which produces, as we have experimentally seen, quite large discrete vector fields, a desirable property for these objects. The latter step, the process to reduce the matrices, is already specified in Coq, but the proof of its correctness is still an ongoing work.

The suitability of our approach has been tested with several examples coming from randomly generated images and also real images associated with a biomedical problem, the study of synaptic density evolution. The results which have been obtained are remarkable since the amount of time necessary to compute homology groups of such images inside Coq is considerably reduced (in fact, it was impossible in the case of biomedical images).

As further work, we have to deal with some formalization issues. Namely, we have to verify that a reduction can be constructed from a matrix and an admissible discrete vector field to a reduced matrix. Moreover, we have hitherto worked with matrices over the ring $\mathbb{Z}_2$; the more general case of matrices with coefficients in a ring $\mathcal{R}$ (with convenient constructive properties) should be studied.

As we have seen in Subsection 4.4, it is necessary the definition of inductive schema which fits to our complex recursive programs. Then, this opens the door to an integration between Coq and the ACL2 Theorem Prover [20]. ACL2 has good heuristics to generate inductive schemes from recursive functions; so, we could translate our functional programs from Coq to ACL2, generate the inductive schemes in ACL2; and finally return such inductive schemes to Coq. Some preliminary experiments have been performed to automate that process, obtaining encouraging results.

In a different research line, we can consider the study of more complex biomedical problems using our certified programs. As an example, the recognition of the structure of neurons seems to involve the computation of homology groups in higher dimensions; a question which could be tackled with our tools.

References

1. Mathematical components team homepage. `http://www.msr-inria.inria.fr/`

Projects/math-components.
2. J. Aransay, C. Ballarin, and J. Rubio. A mechanized proof of the Basic Perturbation Lemma. *Journal of Automated Reasoning*, 40(4):271–292, 2008.
3. G. Barthe and P. Courtieu. Efficient Reasoning about Executable Specifications in Coq. In *Proceedings 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLS'02)*, volume 2410 of *Lectures Notes in Computer Science*, pages 31–46, 2002.
4. M. Bear, B. Connors, and M. Paradiso. *Neuroscience: Exploring the Brain*. Lippincott Williams & Wilkins, 2006.
5. K. Claessen and J. Hughes. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *ACM SIGPLAN Notices*, pages 268–279. ACM Press, 2000.
6. C. Cohen and A. Mahboubi. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination, 2011. http://hal.inria.fr/inria-00593738.
7. Coq development team. The Coq Proof Assistant, version 8.3. Technical report, 2010.
8. G. Cuesto et al. Phosphoinositide-3-Kinase Activation Controls Synaptogenesis and Spinogenesis in Hippocampal Neurons. *The Journal of Neuroscience*, 31(8):2721–2733, 2011.
9. C. Domínguez and J. Rubio. Effective Homology of Bicomplexes, formalized in Coq. *Theoretical Computer Science*, 412:962–970, 2011.
10. X. Dousson, J. Rubio, F. Sergeraert, and Y. Siret. The Kenzo program. Institut Fourier, Grenoble, 1998. http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/.
11. R. Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134:90–145, 1998.
12. G. Gonthier. *Formal proof - The Four-Color Theorem*, volume 55. Notices of the American Mathematical Society, 2008.
13. G. Gonthier and A. Mahboubi. An introduction to small scale reflection in Coq. *Journal of Formal Reasoning*, 3(2):95–152, 2010.
14. P. Graham. *ANSI Common Lisp*. Prentice Hall, 1996.
15. J. Heras, M. Dénès, G. Mata, A. Mörtberg, M. Poza, and V. Siles. Towards a certified computation of homology groups for digital images. In *Proceedings 4th International Workshoph on Computational Topology in Image Context (CTIC'2012)*, To appear in Lectures Notes in Computer Science, 2012.
16. J. Heras, G. Mata, M. Poza, and J. Rubio. Homological processing of biomedical digital images: automation and certification. In *17th International Conferences on Applications of Computer Algebra. Computer Algebra in Algebraic Topology and its applications session*, 2011.
17. J. Heras, V. Pascual, and J. Rubio. Proving with ACL2 the correctness of simplicial sets in the Kenzo system. In *Proceedings of the 20th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'2010)*, volume 6564 of *Lectures Notes in Computer Science*, pages 37–51. Springer, 2011.
18. N. Jacobson. *Basic Algebra II*. W. H. Freeman and Company, 2nd edition, 1989.
19. S. P. Jones et al. The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming*, 13(1):0–255, 2003. http://www.haskell.org.
20. M. Kaufmann and J. S. Moore. ACL2 version 4.3, 2011.
21. L. Lambán, F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. Applying ACL2 to the Formalization of Algebraic Topology: Simplicial Polynomials. In *Proceedings Interactive Theorem Proving (ITP'2011)*, volume 6898 of *Lecture Notes in Computer Science*, pages 200–215, 2011.

22. A. Mörtberg. Constructive algebra in functional programming and type theory. Mathematics, Algorithms and Proofs 2010, 2010. http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/PapersAndSlides.
23. A. Romero and J. Rubio. Homotopy groups of suspended classifying spaces: an experimental approach. *To be published in Mathematics of Computation*, 2012.
24. A. Romero and F. Sergeraert. Discrete Vector Fields and Fundamental Algebraic Topology, 2010. http://arxiv.org/abs/1005.5685v1.
25. J. Rubio and F. Sergeraert. Constructive Algebraic Topology. *Bulletin des Sciences Mathématiques*, 126(5):389–412, 2002.