Final Report for Deliverable Nr. 3.4

Integration with other Proof Systems

Responsible: Jesús María Aransay, jesus-maria.aransay@unirioja.es
Written by: Members of La Rioja Node
Site: Universidad de La Rioja, Spain

Deliverable Date: July 2013

1 Introduction

The task presented in this deliverable is number 3.4, "Integration with other Proof Systems". The task is part of the work package 3, "A Library of Algebraic Topology", in which both the University of Gothenburg and the Universidad de La Rioja are involved. The aim of this task was twofold: first, to develop a set of libraries and formalizations in the field of Algebraic Topology in different proving assistants; second, to investigate the possible communication and transference among these different sources of knowledge. As can be seen in the rest of tasks of the project, the proving assistant which has become the main language of the project has been Coq; consequently, we will not present here the Coq libraries and developments, since they have been already reported in different deliverables of the Formath project; instead, we will focus on the formalizations completed in ACL2 (Section 2.2), Isabelle/HOL (Section 2.1) and also in the developments completed in Coq that are related to some other works in Isabelle/HOL or ACL2. As a second product, we will present also an *ad hoc* tool that we have created to transfer results among different formalisms; the tool has been originally designed to *translate* an Isabelle development to ACL2, but is also capable of translating its input to the OCL language (for formal specifications), and has been also successfully applied to a case study of knowledge transferring from Coq to ACL2. Most of these developments have been submitted (and some of them already accepted) to conferences and journals in their fields. In this deliverable, we present the two aforementioned research lines, and will refer to papers or reports (available from the Formath wiki website, or from the corresponding journals and conferences) for further reading and information.

The report is divided as follows: in Section 2 we sketch stand-alone developments carried out in different proof assistants along the ForMath project; we focus on developments that have not been yet presented in some other reports of the project. In Section 3 we introduce a tool for intercommunication of theorem proving and formal methods tools; in particular, we have built a set of XML libraries that translate developments among different proving assistants, and also to the Ecore + OCL formalism; an intermediary XLL (standing for Small Logical Language) schema has been also created, and some success stories will be also

introduced. Finally, in Section 4 we introduce some conclusions and some of the lessons learned from our work.

2 Developments in different proof assistants

2.1 Isabelle/HOL

One of the crucial tools in the computation of the homology of a (finitely generated) chain complex is the Smith Normal Form algorithm; the algorithm builds a diagonal matrix, in such a way that the elements of the diagonal are either zeroes, or sorted *wrt* a division criterion. The elements in the diagonal are called *invariant factors*. These invariant factors are related to the *dimension* of the homology of the chain complex in each dimension.

Being one of the crucial algorithms in the computation of the homology of chain complexes, we considered appropriate to implement this algorithm in the Isbelle/HOL system; this formalization shall be used as a comparison at various degrees with the already provided Coq formalization. First, the amount of effort invested in each system will give us feedback about the already existing library on matrices for each of them. Additionally, the representation of data structures and Linear Algebra among systems can be also explored. The formalized algorithm should give place also to a piece of executable (formalized) code, whose performance can be also compared for the Coq and Isabelle/HOL versions.

In our Isabelle development we based our work on an existing formalization of matrices applied in the Flyspeck project [19] aimed at the formalization of the proof of the Kepler conjecture. Matrices are represented by an abstract type of functions over pairs of natural numbers (representing the row and column of a matrix) which range is a finite number of elements. The Isabelle code generator facility enables code generation of Isabelle specifications (recursive definitions, constants, equational lemmas, type classes and some operations over sets) to different programming languages, but the abstract type of matrices cannot be directly *code generated* (there is no *equivalent* type in the target languages, for instance SML) and therefore we have to apply a data type refinement to some type that can be suitably represented in the target programming language; the chosen representation was *dense lists of lists of elements* (indeed, a rather familiar type for functional languages), making use of some of the previous results available in [22].

In this Isabelle development we proved that every matrix can be converted to a diagonal matrix by performing *elementary operations* over matrices (interchanging rows or columns, adding to a row another row multiplied by a constant, identically with columns, and multiplying a row or a column by a constant); this development itself summed up a rather huge number of lines, showing that a better infrastructure should be developed. Additionally, we neither have suitable tools for relating matrices to linear forms among modules, and therefore we decided to move backwards and look for a more suitable setting. Our chosen setting was the HOL-Multivariate Analysis session (part of the standard Isabelle library); in this setting, a considerable amount of results about finite-dimensional vector spaces, and specially about *real* vector spaces, can be found. For instance, it is already proved the relation among matrices and linear forms over real vector spaces. The representation of vectors in this setting is introduced by means of functions over a *finite* underlying type (a *matrix* then is represented

as an iterated construction, as a vector of vectors). As an example of the adequacy of this setting for proving results in Linear Algebra, J. Divasón and J. Aransay formalized a proof of the *Rank Nullity theorem* [2].

This representation of vectors and matrices should be, a priori, appropriate for *code generation*, since enumerated types are part of SML. Nevertheless, some operations used in algorithmic Linear Algebra (vector multiplication, matrix multiplication, vector traversing) pose additional demands over the types used to represent the indexes of vectors and matrices. Some technical details and proofs had to be added to the standard Isabelle library before getting to *execute* (both inside of Isabelle, and also by means of the code generation facility, producing *SML code*) this representation of matrices. Additionally, we have developed an Isabelle library in which elementary operations over matrices, and the proofs that these operations represent changes of bases, are introduced; a certain amount of previous results was required, but this infrastructure eases now the implementation of *algorithms* over matrices which are based on exclusively performing these elementary operations; for instance, we have already formalized the correctness of an algorithm computing the Gauss Jordan form (for non square matrices), in which its mathematical meaning (preserving the rank of a matrix representing a linear form) is proved, and the number of lines of this formalization is much smaller than the one of the Smith Normal form formalization in the previous setting. The sketch of the proof and the setup to obtain code generation are available in a separate report [4].

As part of this library, we have also developed a data type refinement for matrices (from functions over finite domains to immutable arrays) which outperforms the representation by means of functions; this data type refinement is also a *product* that can be used in later developments of the same kind (for instance, computation of determinants, inverse matrices or eigenvalues). A report on the performance of such refinement for various fields ($\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}_2$) is available in [3].

2.2 ACL2

Different developments have been also carried out along the ForMath project on the formalization of Homological Algebra computations and digital image processing with the theorem prover ACL2 [5].

In [6] J. Heras *et al* verify the Common Lisp definitions of *simplicial sets* in the Computer Algebra systems EAT [8] and Kenzo [7] (devoted to Homological Algebra computations). Simplicial sets are the *basic constructions* over which higher-order functional operations are performed (Cartesian products, loop spaces etc.), before computing the homological properties (for instance, their homology groups in any dimension) of such spaces. The verification carried out in ACL2 consists in introducing a mathematical definition of simplicial sets, and then proving that the definitions used in both Kenzo and EAT are instances of such definitions; consequently, even if the definitions in the Computer Algebra systems introduce their own computational machinery, they preserve their mathematical properties. Therefore, the certification ensures that the input of the algorithms in EAT and Kenzo are *correct*.

Simplicial polynomials are sums of compositions of *face* and *degeneracy* maps of simplicial sets; they are a useful tool in the proof of the *Normalization Theorem*, a relevant result in

Homological Algebra since it demonstrates the possibility of defining a *reduction* from the chain complex of a simplicial set to the *normalized* version of such a chain complex; usually, the normalized chain complex is *smaller* (in terms of number of generators) and therefore computations over it are faster than in the original one. In [10] L. Lambán *et al* prove in ACL2 the *Normalization Theorem*, based on a previous development of simplicial polynomials and their operations. The work in [9] is a preliminary version of the previous one.

Applications of Algebraic Topology to digital imaging have been also one of the goals of the project (and are presented in a different report, as the Workpackage 3.3). One of the case studies in ACL2 of such applications is found in [11]. In this work, J. Heras *et al* develop a set of basic functions to build simplicial complexes from digital binary images, as a new module of the Kenzo system. Moreover, the correction of these constructing algorithms is proved by means of ACL2, verifying the very same algorithms that are used in the Kenzo system. Further work on digital image processing has been carried out based on this previous development (and is presented in the report of the previous Workpackage).

L. Lambán *et al* present another theoretical development in ACL2 in [12]. The paper presents the formalization of the so-called *Eilenberg-Zilber* Theorem; this result claims that, given two simplicial sets, there exists a reduction from the chain complex generated from their Cartesian product to the tensorial product of the chain complexes associated to each of them. The result is of theoretical interest, from the mathematical point of view, but it is also computationally relevant (for instance, it is heavily used in the Kenzo system) since the result states the explicit definition of the morphisms that define the reduction. The formalization is quite intricate; it makes an intensive use of previously developed tools, such as simplicial polynomials (extended to the bivariate case, also by L. Lambán *et al* [13]), and still sums up more than 13000 ACL2 code lines. Interestingly, the algorithmic nature of the theorem enables the comparison of the ACL2 verified statement of the theorem with the algorithm performing this operation in the Kenzo system, allowing us to test that both algorithms are equivalent over numerous meaningful examples (even if the Kenzo original version is still preferred because of its superior performance).

3 Interoperation among systems

In this section we present the *I2AE* (standing for Isabelle to ACL2 and Ecore) tool, which is used to interoperate among different widespread formal method tools. The tool, developed by J. Divasón *et al* is presented at large in [14]; here we will describe our aim at its development, as well as the different parts of the tool and the languages that have been used along its development 3.1. In Section 3.2 we present three different case studies that we have developed using the interoperability tool, which show its adequacy for certain purposes. The tool has been also presented in two different papers, which have been submitted for publication [15, 16].

3.1 A tool for interoperation among formal methods tools

After completing a formalisation in Isabelle/HOL of an algorithm computing a diagonal matrix, which is commonly used as a previous step in the computation of the Smith Normal Form of a matrix (see Section 2.1 for details), we considered the possibility of *reusing* parts

of this work in a similar formalisation in ACL2. The reasons to *port* our development from Isabelle to ACL2 are mainly two; first, ACL2 is a programming language, a subset of Common Lisp, and therefore an algorithm implemented and formalized in ACL2 can be executed without further manipulations or refinements. Second, *Kenzo* is written in Common Lisp, and contains its own version of the algorithm computing the Smith Normal Form, which could be replaced, or at least compared (by means of its results, in the spirit presented in [12]) to the one being formalized.

The tool has been built for this particular case study, but we have almost exclusively applied XML transformations and made use of a generic XML schema (XLL, standing for *Small Logical Language*) in such a way that translations can be also ported to other theorem provers (such as Coq, as we will show in Section 3.2), and also to other formal tools, such as Ecore models (which are themselves formed by both a set of UML classes and OCL logical restrictions operating over the classes), as we present in this section.

It is worth noting that the translations are focused on syntactic matters, and therefore we propose a *shallow embedding* among the different systems. We translate among the systems the data types identifiers, the arity of operations or definitions over them, and the statements (not the proof objects) of the lemmas leading to a complex result. These ingredients, that could be seen as a *proof plan*, are translated to XLL and then translated again to the pretended formalism. Theoretical issues show up; for instance, Isabelle/HOL is more expressive than ACL2 or OCL (both of them based on first-order logics), but our case studies, which represent significative developments (more than 6.000 Isabelle code lines) fitted in the logical language proposed in XLL, and therefore can be translated to ACL2 and OCL.

In the following we present a description of the tool, together with the different formalisms that are involved in the translations. Our starting point, as we said above, is a given development in Isabelle/HOL; from it, a set of transformations is defined that enable us to transfer the types, specification of definitions and theorems statements to both ACL2 and to an Ecore model (given by UML class definitions and OCL restrictions). The translations to Ecore and ACL2 are both based on an intermediary XLL document, automatically generated from the Isabelle sources. XLL includes elements for specifying types, operations and logical statements relating them; its syntax can be divided into two substantial parts; one of them is used to state the definitions of data types and its operations appearing in the input Isabelle theory, whereas the second part contains the statements of the properties included in the same theory; the XLL syntax is presented in the extended report [14]. The XML schema of XLL performs additional operations to ensure that the statements are related to types and operations that have been previously introduced (*i.e.*, that the specification of properties corresponds with elements that have been also introduced in the generated XML).

The XLL language is used as a common basis to perform translations to both Ecore and ACL2 of the original Isabelle developments (or *theories*, in the Isabelle jargon, even when an Isabelle theory can include additional constructs not supported in XLL, such as type classes or locales).

The translation to Ecore, where OCL is also a first order language, serves as a general purpose formal specification of the theory we are carrying out. The translation includes the definition of data types and the specification of properties; both are made starting from the XLL file of each theory; the translation of the data types in XLL gives place to the

Ecore (UML) classes (or the "context") over which the OCL properties have to be stated; the translation of the statements in XLL produces OCL properties, which make reference to the generated Ecore classes and its operations. Additionally, this specification of properties and data types could be used to build (automatically or provided by the user, depending on the underlying technologies) instances of the Isabelle specifications, where the properties (the theorems statements) could be thoroughly checked or refuted, (once the Isabelle specifications have been proved, and considering the translation as correct, faults could be only originated in the representation chosen for the instances, guiding us in the process of choosing a suitable representation; if the Isabelle properties have not been proved, the Ecore models could be considered as a testing tool for the properties).

Then, the translation from XLL to ACL2 shows that the formal specification of the theory can be ported to some other theorem proving tools. The reason why we have chosen ACL2 is that we are familiar with its syntax, its simplicity (apart of the particular treatment applied to existential and universal quantifiers, that we introduce later) and also our personal interest in transporting the theory about matrices that we have completed in Isabelle to it. Nevertheless, there is nothing inherent to ACL2 that is needed to carry out the translation, so probably a similar transformation could be carried out to some other theorem provers without a substantial effort. As we have said before, the expressivity of HOL is larger than the one of ACL2, which means that our use cases will restrict to their common subset. These tools (ACL2, in our case) now could use the statements transferred from the Isabelle formal development as a guideline to achieve a similar formalization to the original one.

In the following diagram we present the different steps that are carried out to produce the translation. These steps are briefly explained below; the report in [14] includes a more detailed description.

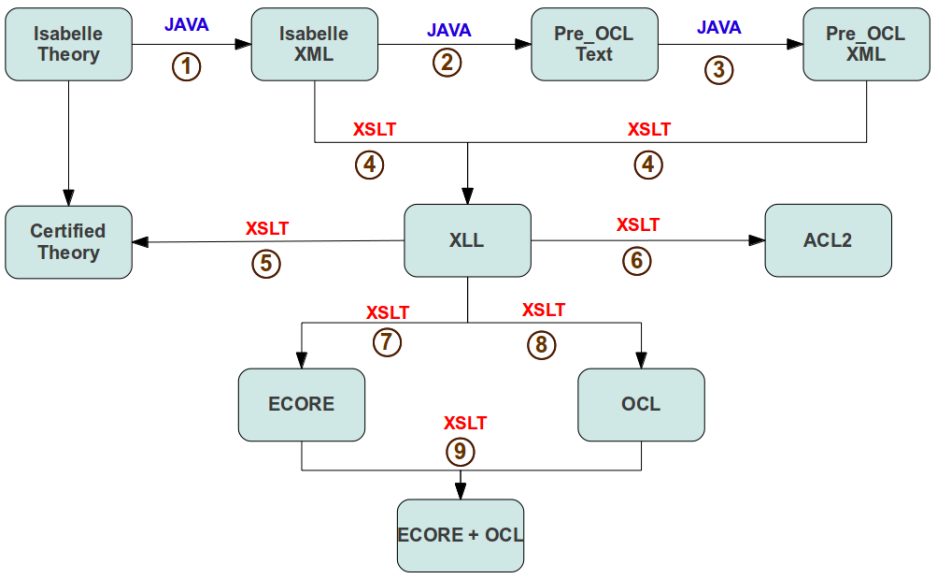


Figure 1: Architecture of our development

The previous steps are exclusively developed using three technologies: Java, XSLT and XProc. As it can be observed in the image, step 5 includes an additional certification of the logical statements translated to XLL being *provable equivalent* to the original ones in the Isabelle theory; this increases the confidence in the translations carried out (or could show possible flaws in the translation process to be fixed modifying the translation process).

The crucial transformation in the architecture presented in Figure 1 is the translation from an Isabelle input to an XLL file (through the steps 1 to 4); in these steps, an intermediary XML language called *Pre_OCL* is created to express the statements of properties; in order to formally prove that this transformation can be reversed, step 5 proves that the statements presented in the XLL document generated through the steps 1 to 4 are equivalent to the original Isabelle statements (inside of Isabelle). Step 6 translates the statements presented in the XLL to ACL2; these statements should provide the guideline to formalize the original Isabelle theory in ACL2. Steps 7, 8 (which could be understood as a single step) and 9 translate the XLL specification of types and properties to the Ecore + OCL environment.

The translation process works reasonably well for theories which do not make use of higher-order statements. It seems rather possible that, for a given Isabelle theory (restricted to first-order constructions), the XSLT transformations could be minimally adapted in order to properly translate that theory to ACL2 and Ecore with OCL restrictions through an XLL document. Since our starting point are Isabelle theories, whose type system is rather simple (enumerated or recursive types, functions and product types) and the output of our technology is Ecore, which is a modeling language with a richer type system, delicate design decisions should be made. Contrarily, ACL2 and OCL are based on first-order predicate logic, whereas Isabelle is based on HOL (higher-order logic). Thus, ACL2 and OCL are less expressive and the translation of statements could give place to XLL files that would not be compliant to the XLL schema (and therefore would not produce any statements in the target languages).

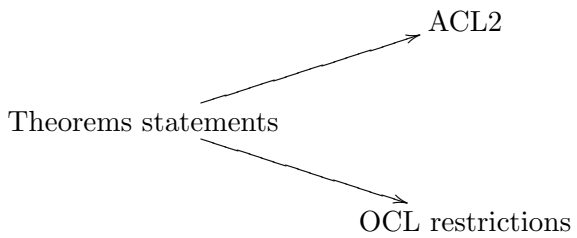
In order to get the ACL2 output, we will be mainly interested in the theorems statements appearing in the XLL file. ACL2 is untyped, thus to introduce operations in ACL2 we only need to know its arity, which can be directly inferred from the original statements or definitions. Nevertheless, ACL2 usually introduces predicates to state that free variables or constants appearing in operations and formulas belong to the appropriate sets (for instance, that an index for a list or a matrix position is a natural number). These predicates are usually known as *recognisers*; they are also generated by our automatic translation. It is worth noting that these recognisers are necessary for every argument appearing in a function (including the output of the function). This kind of information is automatically included in Isabelle, because the type system enforces the input and output values of a function to be of a concrete type, but has to be labelled *manually* in ACL2 (for instance, by means of *recognisers*). We detected (and repaired) this weakness of our translation tool after carrying out experiments over Isabelle developments.

Data types representation is not translated from Isabelle to ACL2 since we prefer to preserve the internal design decisions of ACL2 to this language. As we have already mentioned, our idea is to translate a *proof plan* from Isabelle to ACL2, but it is preferable to keep the internal representation issues inside of each theorem prover; ACL2 and Isabelle are based on different type systems and promote different representations (as we will show in Section 3.2,

our representation of matrices in Isabelle and ACL2 are rather different).

As a crucial difference between ACL2 and Ecore, modeling in the latter requires information about data types (in order to convert each type to a class) and about the types and arity of functions/definitions used in Isabelle (in order to convert them to class methods). On the contrary, as we already pointed out in the translation to ACL2, we obtain an Ecore model, so the information about implementation or representation from Isabelle is not required for this transformation. To complete the Ecore model, we also translate Isabelle theorems statements to OCL restrictions in that model; it is worth noting that XLL imposes additional restrictions on its logical statements, since they can only refer to types and operations that are also stated in the XLL part describing data types. This additional validations enforce that the XLL document is *self-contained*, in the sense that types, operations and statements only make reference to elements in the very same XLL document.

The following diagrams show which parts of an Isabelle development are used to generate each part of the target languages specification:



Type definitions, operations $\longrightarrow$ Ecore MODEL

The Isabelle theory will be translated to ACL2 and Ecore + OCL by means of XLL, trying to minimize the "ad-hoc" translations (step 6 or steps 7 to 9). Firstly, some XML files are generated with the necessary information, that is, information about types (necessary for modeling in Ecore) and information about theorems' statements (necessary for translating them to ACL2 and OCL). This first step is capable of solving dependencies on the types appearing on operations (for instance, if we are to generate the type of matrices over integer numbers, the Isabelle integer type will be also translated to XLL). It also performs the necessary operations to generate in XLL the basic logical operations that appear in Isabelle statements (such as $\wedge$, $\vee$). The tool also deals with universal and existential quantifiers in Isabelle statements, translating them to appropriate predicate functions in XLL. The translation process to ACL2 will take care of mapping these existential and universal operations to the adequate ACL2 operators (ACL2 uses *Skolemization* to introduce quantifiers, since they are not part of its original foundations). Once all this information is translated into XML files, the Isabelle theory is no longer necessary; we just keep it to complete some further tests about the relationship between the generated statements in some of the intermediary languages developed (Pre_OCL and XLL) and the original theorems. One of the key products in the development process is the Pre_OCL file (step 3). It contains an XML dialect with the theorems statements from the input Isabelle theory, but now written in a "raw" Isabelle style, where operations appear in prefix notation, most of the abundant Isabelle syntax translations

are avoided (for instance, Isabelle pretty syntax for lists $[x, y]$ would appear in terms of the primitive Isabelle list constructors), and additionally following the XML structure presented in [14, Appendinx 1]. This Pre_OCL XML file is completed with the specification of the datatypes used in the Isabelle theory (typedefs, definitions, functions...) to obtain the XLL document, which is the keystone of our development.

3.2 Success stories

I2AE, the tool presented in the previous section, has been used in different experiments of translating developments in theorem provers to different formalisms. These experiments have proved the suitability of the tool to its aim, and also have provided valuable feedback that helped us to improve some functionalities of the tool.

Our first example, developed by J. Divasón *et al*, was presented mainly as a *tutorial* on how to use the developed XML and Java translations; it is included in the report presenting the tool [14]. The experiment consists on an example introducing the *list* type, and then some basic operations and properties of lists. Interestingly, the Isabelle type of lists is introduced over a variable type α, which in the Ecore model will give place to a parameterized class, whereas in ACL2 it will not be translated (just a corresponding *recogniser* will be introduced in the system).

The remaining part of the example requires manually selecting the Isabelle elements that will be generated (types, operations and lemmas) and applying the automatic translations that have been shown in Figure 1. The Isabelle input gives place to an ACL2 collection of theorems (that should be proved by the user) and also to an Ecore model (both some UML classes, one representing lists, another one representing the parametric type and its required operations and a collection of OCL restrictions over the previous classes).

Being the Isabelle and ACL2 definitions of the type *list* inductively defined types (by means of the empty list and a *cons* operation), in this example the theorems proved in Isabelle showed its utility as a *proof plan* in ACL2, helping us to complete a similar development in this theorem prover.

The second example, carried out by J. Divasón *et al*, is quite more challenging. It has been described in [15], and briefly sketched here. The interoperability setting is applied to an Isabelle/HOL formalisation of some well known results about integer matrices. It is important to highlight that, even if the input theory is written in HOL, the problem is essentially of a first-order nature, and therefore the information that is lost when going from Isabelle/HOL to XLL does not prevent us from getting a sensible specification. Thus, we consider an Isabelle/HOL development which defines a verified method to reduce a given matrix to a diagonal form; i.e. a method to compute a diagonal matrix which is *equivalent* to the initial one – let us recall that two matrices A and B are equivalent if there exist two invertible matrices P and Q such that $B = PAQ$.

Then, the corresponding Isabelle/HOL formalisation includes the basic matrix operations (addition and multiplication) and properties of the ring of integer matrices. The main result of this Isabelle/HOL theory can be expressed as the following result:

Given an integer matrix A, there exist three integer matrices P, Q and B such that:

- $B = PAQ$.

- *P and Q are invertible matrices.*
- *B is a diagonal matrix.*

The diagonal matrix presented in the previous lemma is usually computed in many algorithms as an intermediary step in the computation of the Smith Normal Form (see [17, 18]); indeed, this particular matrix, has its own interesting properties.[1]

Prior to describing the translation, we briefly describe the Isabelle/HOL formalisation that leads us to prove the previous result.

One of the most relevant decisions in the initial steps of a formalisation is the choice of a suitable representation for the objects involved in the development; in this particular case, integer matrices. In our Isabelle/HOL theory, the family of matrices is represented as the set of functions with two arguments of type `nat` and finitely many non-zero positions. This functional representation eases the definition of operations over matrices and the proof of properties (indeed, it is part of the Isabelle distribution and some preliminary results are already available). The formal definition is:

```
definition nonzero_positions ::
        "(nat ⇒ nat ⇒ 'a::zero) ⇒ (nat x nat) set" where
        "nonzero_positions A = {pos. A (fst pos) (snd pos) ~= 0}"

definition "matrix = {(f::(nat ⇒ nat ⇒ 'a::zero)).
                      finite (nonzero_positions f)}"

typedef 'a matrix = "matrix :: (nat ⇒ nat ⇒ 'a::zero) set"
```

The library offers several definitions and properties over this data type; in particular, operations *nrows* and *ncols* that, making use of the Hilbert's ϵ operator, return the maximum row and column which contain nonzero elements. We had to define elementary operations on matrices; in particular, there are two basic operations for our development (*interchange_rows_matrix* and *row_add_matrix*) which allow us to exchange two rows of a matrix and replace a row by the sum of itself and another row multiplied by an integer (the corresponding operations acting on columns are also defined). We introduce here the former, as well as the definition of its *functional* behaviour over the *underlying* representation of matrices (in this case, *functions*, see above):

```
definition interchange_rows_infmatrix ::
        "int infmatrix ⇒ nat ⇒ nat ⇒ int infmatrix"
  where "interchange_rows_infmatrix A n m ==
      (λi j. if i=n then A m j else if i=m then A n j else A i j)"
```

[1] In spite of the fact that the calculation on the Smith Normal Form is a more renowned result than the one presented, the diagonal form is enough in many calculations. For example, the homology of a chain complex over a ring can be obtained using the diagonal form of the differential maps represented as matrices. This situation is usual in some programs for Symbolic Computation in Algebraic Topology; thus, the presented result has its own interest in that area.

```
definition interchange_rows_matrix ::
       "int matrix ⇒ nat ⇒ nat ⇒ int matrix"
  where "interchange_rows_matrix A n m==
       Abs_matrix (interchange_rows_infmatrix (Rep_matrix A) n m)"
```

The previous definition relies on the type morphisms `Abs_matrix` and `Rep_matrix`, which perform the conversion between the type (`matrix`) and the underlying type (`infmatrix`, an abbreviation of the functional representation of matrices). It makes use of the function `interchange_rows_infmatrix`, which represents the functional behaviour of the elementary operation.

Using these functions (and their column counterparts), we can define several auxiliary results and finally state and prove the previous result about matrices in Isabelle/HOL.

```
lemma Diagonalize_theorem:
shows "∃P Q B. is_invertible P ∧ is_invertible Q ∧ B = P*A*Q
  ∧ is_square P (nrows (A::int matrix)) ∧ is_square Q (ncols A)
  ∧ Diagonalize_p B (max (nrows A) (ncols A))"
```

The proof of this result (from a conceptual point of view) is *constructive*; that is, the witnesses (P, Q and B in this case) for the existential expression (it applies the existential quantifier to P, Q and B) are explicit and could be algorithmically produced. Roughly speaking, the procedure used in the proof to compute the equivalent diagonal matrix is analogous to the Gauss-Jordan elimination method.

The fact that the essence of the proof is constructive would allow us to obtain executable programs from it. However, it is not possible to directly execute the corresponding expressions inside Isabelle due to the representation of matrices (based on the abstract type *matrix*). It is well-known that, in such a situation, we could *refine* the data structure representation (by taking, for instance, lists of lists to represent matrices), in order to get executable code from Isabelle. Our approach is different: we look for a proof in ACL2, where executability will be guaranteed.

As a first step in the translation of the Isabelle/HOL theory to ACL2, the corresponding XLL document – an XML instance compliant with the XLL schema – is generated following the steps presented in Fig. 1. The XLL description of the theory consists of two different components: data types and logical statements.

The former contains the specification of the data types appearing in the source Isabelle/HOL theory (their names and the collection of functions where the types appear as parameters, which are selected by the tool). This information is automatically organized in a *class*, an XLL structure which is used to represent each type and its operations (and that in our Ecore experiments is later assigned to a UML class), whose XLL description (for the type `matrix`) appears in Figure 2 (we only include elementary operations over matrices).

The second component of the resulting XLL document consists of a set of statements establishing the properties of the entities (data and methods) involved in the theory. To illustrate this component, we include in Figure 3 the XLL description of the lemma which states in Isabelle/HOL the idempotence property of `interchange_rows_matrix` – the square of this function is the identity.

```
<Class name="Matrix.matrix">
  <Class_Parameters>
    <Parameter name="alpha">
      <Type name="Int.int"/>
    </Parameter>
  </Class_Parameters>
       ...
  <method name="Diagonal_form.interchange_rows_matrix">
    <Type name="Matrix.matrix"/>
    <Input name="n"><Type name="Nat.nat"/></Input>
    <Input name="m"><Type name="Nat.nat"/></Input>
  </method>
       ...
  <method name="Diagonal_form.row_add_matrix">
    <Type name="Matrix.matrix"/>
    <Input name="i"><Type name="Nat.nat"/></Input>
    <Input name="j"><Type name="Nat.nat"/></Input>
    <Input name="q"><Type name="Int.int"/></Input>
  </method>
  ...
</Class>
```

Figure 2: XLL for the generated `matrix` class.

```
lemma interchange_rows_matrix_id:
 shows
 "interchange_rows_matrix (interchange_rows_matrix A n m) n m = A"
```

From the XLL description of the Isabelle/HOL theory we obtain an ACL2 specification. For instance, the ACL2 functions obtained from the XLL file of Figure 2 are:

```
(defun Diagonal_form.interchange_rows_matrix (A n m )
    (declare (ignore A n m ))
    nil)

(defun Diagonal_form.row_add_matrix (A i j q )
    (declare (ignore A i j q))
    nil)
```

It is worth noting that the body of these functions is empty (in fact, the value nil is returned by default to allow the compilation of the function); – note that data types are very linked to the way of working in each proof assistant. Therefore, it is unlikely that the Isabelle/HOL representation of matrices will be the most useful to work in ACL2. Then, we delegate the task of defining a suitable representation of the data types to a further step in the development process.

In the same way, theorems like the one presented in Figure 3 are also translated to ACL2.

```
(defthm interchange_rows_id
    (implies
      (and (matrix_integerp A) (natp n) (natp m))
      (equal (Diagonal_form.interchange_rows_matrix
           (Diagonal_form.interchange_rows_matrix A n m) n m) A)))
```

Using this procedure, we translate the whole Isabelle/HOL development into ACL2. The ACL2 version of diagonalization result is stated as follows.

```
(defun-sk exists_Diagonalize_theorem (A)
  (exists (P Q B)
       (and (Diagonal_form.is_invertible P)
       (and (Diagonal_form.is_invertible Q)
       (and (equal B (Groups.times_class.times
                     (Groups.times_class.times P A) Q))
       (and (Diagonal_form.is_square P (Matrix.nrows A))
       (and (Diagonal_form.is_square Q (Matrix.ncols A))
       (Diagonal_form.Diagonalize_p B
          (max (Matrix.nrows A) (Matrix.ncols A)))))))))

(defthm Diagonalize_theorem
  (implies (matrix_integerp A)
           (exists_Diagonalize_theorem A)))
```

```
<Theorem>
    <name>interchange_rows_id</name>
    <forall>
      <param>
        <name>A</name>
        <type>Int.int Matrix.matrix</type>
      </param>
      <body>
        <forall>
          <param>
            <name>n</name>
            <type>Nat.nat</type>
          </param>
          <body>
            <forall>
              <param>
                <name>m</name>
                <type>Nat.nat</type>
              </param>
              <body>
                <operation>
                  <name>HOL.eq</name>
                  <operation>
                    <name>Diagonal_form.interchange_rows_matrix</name>
                    <operation>
                      <name>Diagonal_form.interchange_rows_matrix</name>
                      <constant> <name>A</name> </constant>
                      <constant> <name>n</name> </constant>
                      <constant> <name>m</name> </constant>
                    </operation>
                    <constant> <name>n</name> </constant>
                    <constant> <name>m</name> </constant>
                  </operation>
                  <constant> <name>A</name> </constant>
                </operation>
              </body>
            </forall>
          </body>
        </forall>
      </body>
    </forall>
  </Theorem>
```

Figure 3: XLL for the `interchange_rows_id` theorem.

The previous statements, transferred from the Isabelle/HOL formal development, can be used as a guideline to achieve a similar formalisation in ACL2. Indeed, ir our experiments in ACL2 [15], we have observed that the Isabelle proof plan is useful to prove properties of the algorithm in ACL2. Additionally, the ACL2 proofs demanded from us some additional lemmas (more concretely, recognisers in ACL2 resembling the Isabelle types of the parameters and the output of functions) that will be automatically generated by the infrastructure in future releases, enhancing its utility.

The very same development has been also translated to Ecore, giving place to a collection of UML classes and OCL restrictions. The translation process has been fully automatic. More details of the translation are given in [16].

There is a third case study in which the I2AE tool has shown its usefulness [20], developed by J. Heras *et al.* In this problem there are additional tools that ease a *multi-tool strategy* to program verification. The goal of this work consists in verifying some Java algorithms that perform computations on digital images. These computations are related to *crucial properties* of the digital images, such as their number of connected components, whose computation *by hand* amounts to a great workload for biologists. Therefore, developing the programs that ease this task and verifying them is a sensible goal. More details on this research are given in the report of Workpackage 3.3, but here we focus only on one experiment which emphasizes the integration of different theorem provers to increase the computations reliability.

In [20], starting from Java programs, the Why/Krakatoa tools [21] are applied to verify the JML annotations that have been introduced in the Java programs. Krakatoa is in charge of reading the annotated Java files and producing a representation of the semantics of the Java program into Why's input language. *Why* computes proof obligations for a core imperative language annotated with pre- and post-conditions; then, various automated theorem provers are used to prove the proof obligations. If any of the previous obligations cannot be automatically discharged, they are translated to Coq, so that the user can try to prove the missing properties in this proof assistant. At the end of the process, every proof obligation should have been discharged.

In some circumstances, one can fail to directly generate the Coq proofs. Here is where the I2AE tool again proves its usefulness; with a new XML translation, which produces XLL code starting from Coq, ACL2 statements are generated; the user provides ACL2 definitions for the operations involved and then tries to see whether ACL2 can complete such proofs, or what kind of hints are required in the process. Basically, the idea is to introduce an additional *semiautomated* theorem prover, ACL2, to discharge proof obligations that would have to be interactively proved in Coq. In the work presented above, this methodology shows its usefulness helping to complete some proofs in the multi-tool strategy by means of ACL2.

The work also succeeds to verify, following this multitool approach, some Java snippets that are intensively used in the preprocessing of biological digital images captured with confocal microscopes.

4 Conclusions

The integration of theorem provers is a daunting task; there are several works, conferences, workshops and seminars that face this subject from very different points of view (ranging from the logical foundations of different systems and their relationships to particular case studies giving place to shallow or deep embeddings, and also the definition of metalanguages that enable the communication among different theorem provers).

Our proposal was focused on the different theorem proving tools that have been used along the ForMath project (Coq, Isabelle and ACL2), and more concretely, in the developments that have been carried out with them along the project (formalisations of Algebraic Topology results and algorithmic Linear Algebra). In this voluntarily limited scope, the results presented in this report show that we have achieved some interesting results, both in the formalisation of results and also in the integration of theorem provers to interchange these results. Additionally, the tools developed and the results obtained rely on different Workpackages of the ForMath project.

References

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

[2] J. Aransay and J. Divasón. Rank Nullity Theorem in Linear Algebra. Archive of Formal Proofs. 2013.

[3] J. Aransay and J. Divasón. Performance Analysis of a Verified Linear Algebra Program in SML. Technical report available at http://wiki.portal.chalmers.se/cse/uploads/ForMath/pavlap. 2013.

[4] J. Aransay and J. Divasón. Formalization and execution of Linear Algebra: from theorems to algorithms. Technical report available at http://wiki.portal.chalmers.se/cse/uploads/ForMath/felafte. 2013.

[5] M. Kaufmann, P. Manolios and J S. Moore. Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers. 2000.

[6] J. Heras, V. Pascual and J. Rubio. Proving with ACL2 the correctness of simplicial sets in the Kenzo system. LOPSTR11, Lecture Notes in Computer Science 6564, pp. 37–51, 2011. Springer.

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

[8] J. Rubio, F. Sergeraert and Y. Siret. EAT: Symbolic Software for Effective Homology Computation. Institut Fourier, Grenoble, 1990. http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/#Eat.

[9] L. Lambán, F. J. Martín-Mateos, J. L. Ruiz-Reina and J. Rubio. Applying ACL2 to the Formalization of Algebraic Topology: Simplicial Polynomials. ITP11, Lecture Notes in Computer Science, 6898, pp. 200–215, 2011. Springer.

[10] L. Lambán, F. J. Martín-Mateos, J. L. Ruiz-Reina and J. Rubio. Formalization of a Normalization Theorem in Simplicial Topology. Annals of Mathematics and Artificial Intelligence 64(1): 1 – 37. 2012.

[11] J. Heras, V. Pascual and J. Rubio. A Certified Module to Study Digital Images with the Kenzo System. EUROCAST2011, Lecture Notes in Computer Science 6927, pp. 37–51, 2011. Springer.

[12] L. Lambán, F. J. Martín-Mateos, J. L. Ruiz-Reina and J. Rubio. Verifying the bridge between Simplicial Topology and Algebra: the Eilenberg-Zilber algorithm. Accepted for publication in Logic Journal of the IGPL.

[13] L. Lambán, F. J. Martín-Mateos, J. L. Ruiz-Reina and J. Rubio. Certified Symbolic Manipulation: Bivariate Simplicial Polynomials. ISSAC2013, ACM Press, pp. 243 – 250. 2013.

[14] J. Aransay, J. Divasón, J. Heras, L. Lambán, V. Pascual, A. L. Rubio and J. Rubio. A report on an experiment in porting formal theories from Isabelle/HOL to Ecore and OCL2. Technical Report. 2013. http://wiki.portal.chalmers.se/cse/uploads/ForMath/isabelle_acl2_report

[15] J. Aransay, J. Divasón, J. Heras, L. Lambán, V. Pascual, A. L. Rubio and J. Rubio. Transforming (almost) without programming: a case of interoperability among theorem provers. Technical Report. 2013. http://wiki.portal.chalmers.se/cse/uploads/ForMath/twpcitp

[16] J. Aransay, J. Divasón, J. Heras, L. Lambán, V. Pascual, A. L. Rubio and J. Rubio. Obtaining an ACL2 specification from an Isabelle/HOL theory. Technical Report. 2013. http://wiki.portal.chalmers.se/cse/uploads/ForMath/oafsit

[17] G. H. Bradley. Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations. Mathematics of Computation 25 (116): 897 – 907. 1971.

[18] H. Cohen. A Course in Computational Algebraic Number Theory. Springer, 1995.

[19] S. Obua and T. Nipkow. Flyspeck II: the basic linear programs. Annals of Mathematics and Artificial Intelligence 56 (3–4): 245 – 272. 2009.

[20] J. Heras, G. Mata, A. Romero, J. Rubio and R. Sáenz. Verifying a platform for digital imaging: a multi-tool strategy. CICM 2013. Lecture Notes in Computer Science, 7961, pp. 66–81, 2013. Springer.

[21] J. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus Platform for Deductive Program verification. CAV 07. Lecture Notes in Computer Science, 4590. pp. 173 – 177, 2007. Springer.

[22] C. Sternagel and R. Thiemann. Executable Matrix Operations on Matrices of Arbitrary Dimensions. Archive of Formal Proofs. 2010. http://afp.sf.net/entries/Matrix.shtml