TESIS DOCTORAL

Razonamiento mecanizado en álgebra homológica

Jesús María Aransay Azofra

UNIVERSIDAD DE LA RIOJA

TESIS DOCTORAL

Razonamiento mecanizado en álgebra homológica

Jesús María Aransay Azofra

Universidad de La Rioja
Servicio de Publicaciones
2006

Esta tesis doctoral, dirigida por los doctores D. Julio Rubio García y D. Clemens Ballarin, fue leída el 4 de abril de 2006, y obtuvo la calificación de Sobresaliente Cum Laude por Unanimidad

Edita: Universidad de La Rioja
Servicio de Publicaciones

ISBN 84-689-8743-3

Razonamiento mecanizado en Álgebra Homológica

Memoria presentada para la obtención
del título de Doctor

Jesús María Aransay Azofra

Directores: Dr. D. Julio Rubio García
Dr. D. Clemens Ballarin

Universidad de La Rioja

Departamento de Matemáticas y Computación

Logroño, Enero de 2006

Mechanized Reasoning in Homological Algebra

Dissertation submitted for the degree
of Doctor of Philosophy

Jesús María Aransay Azofra

Advisors: Dr. D. Julio Rubio García
Dr. D. Clemens Ballarin

Universidad de La Rioja

Departamento de Matemáticas y Computación

Logroño, January 2006

This work has been partially supported by projects DGES (PB98-1621-C02) and SEUI-MEC (TIC2002-01626) from the Spanish Government, project CALCULEMUS (HPRN-CT-2000-00102) from the European Union, project ACPI2002/06 from Comunidad de La Rioja, and project API-02/24 and grants ATUR02/03, ATUR03/03, ATUR04/04 from Universidad de La Rioja.

Acknowledgements

Julio Rubio and Clemens Ballarin made this work possible. Julio, from the nearness. He gave me the chance to introduce me in the research's world. Due to his interest I enjoyed great academic opportunities. In addition to this, his availability to discuss new ideas, questions or simply comments that emerged along the elaboration of the memoir, as well as his cordiality, were always remarkable. His enthusiasm helped me to overcome the difficult moments. Clemens and his attention were always motivating to go ahead. He permanently showed his accessibility and willingness to collaborate in everything I required, despite of the distance during much of this work.

In our research's group at the Universidad de La Rioja, I must acknowledge the work of Laureano Lambán. His words were always encouraging and his advice useful. Vico Pascual and César Domínguez showed me the way I must follow with their great task, and always tried to help me with their experience. My first research works were joint with Juan José Olarte. Ana Romero, Mirian Andrés, Ángel Luis Rubio, Francisco José García and Eloy Mata were there to listen and value each of the ideas introduced here. I take the chance to acknowledge to all the members of Departamento de Matemáticas y Computación at the Universidad de La Rioja for the kindnesses they have always shown to me. I would also like to thank every friend that came into my office with the intention of taking a coffee or talk for a while.

Jacques Calmet gave me the chance to work during six months in the Institüt für Algorithmen und Kognitive Systeme at the Universität Karlsruhe. This research stay helped me to both define the problems which are tackled in this memoir, and, personally, to know how hard and satisfactory research can be.

The research group of Tobias Nipkow at Technische Universität München welcomed me for ten weeks really kindly. Working and living together with them was a greatly enriching experience.

Finally, I must thank my family for believing in me and in everything I did. My parents and my brother Óscar always supported me and trusted me. Judith, with her patience and love, made everything easier and gratifying.

To all of them, thanks.

Contents

Introduction

Mathematics is not a formal science. Or, at least, it is not formal in the strict sense of symbolic logic. Working mathematicians have much freedom with respect to foundational or formal aspects. For instance, when applying a previous result, less effort is devoted to show that every premise holds, except in cases where a condition seems to be difficult to be checked or interesting in itself. Obviously, notions such as *difficulty* or *interest* are quite elusive from a formal point of view.

Nevertheless, this is neither a completely subjective issue. The level of detail seems to be rather a social question which depends on several variables, ranging, among others, from the expected audience (less detail in research notes, more detail in undergraduate textbooks) to the discipline itself (for instance, it is well known that some theoretical physics arguments are not considered rigorous enough from a pure mathematical point of view).

With respect to the logic underlying their reasoning, standard mathematicians are even more sloppy. When dealing with basics, they usually rely on some kind of naive set theory, supposed to be based, in a more or less conscious way, on Zermelo-Fraenkel axiomatic system.

Variables are another source of inaccuracies in standard mathematical texts. Quite often, the type of variables (that is to say, the kind of values a variable can be replaced by) is undetermined or based on a typographical convention. Variables are not always clearly signaled as free or bounded, and in this second case, it is quite unusual to be absolutely precise with respect to the binding scope or range. In this context, it is relatively frequent to rename a variable in some places, but not at every suitable occurrence.

Two consequences of this informal presentation of mathematics, of very different nature (and importance), can be stressed. On the one hand, a relatively great amount of published proofs are not such, because they contain (usually very minor) defects from a formal point of view. On the other hand, this distance from the (boring and tiring) exigencies of symbolic logic allows Mathematics to be developed in an extremely powerful and quick way which can be considered as a characteristic of the field as a whole.

The reason for this behaviour of Mathematics is that mathematicians have reached a high reasoning level, and, even more, they seem to be able to increase in an unlimited way the abstraction level of reasoning.

Why can some degree of inaccuracy live together with the firm evolution of Mathematics? The answer depends on the differences among three related concepts:

- the *idea* underlying a proof;
- the presentation of the proof in a mathematical text;
- the very proof in the sense of symbolic logic.

If an idea is correct, then it is accepted that its presentation can be done in a (more or less) loose way. Of course, the idea is *correct* if it can be converted (in a more or less elaborate manner) into a proof in the formal sense.

It is in the interplay among these three entities that the relevance of *Theorem Provers* (TPs, in the sequel) appears. A TP is a computer program aimed at constructing proofs in a logical sense. From a utopian viewpoint, a TP should be capable of, given an idea of a proof, obtaining the corresponding formal proof in an automated way.

In fact, historically, the goal of the first Theorem Provers was even more pretentious: from a *claim*, the system should automatically refute or prove it, and in this second case, it should find a formal proof of the claim (which then could be considered as a *theorem*). These optimistic hopes were directed by the complete solution of the problem in the propositional case, and also by Robinson's resolution algorithm. They gave rise to the first generation of *Automated Theorem Provers* (ATPs) (see [Davis, 2001] for a historical review on these first systems).

Nevertheless, and despite of the success of systems such as OTTER [McCune, 2003], it was recognized in early stages that a certain human intervention would be necessary, both to design the claim (in the precise formalism of the ATP[1]) and to provide guidelines for the proof. In the most automated systems, such as OTTER or ACL2 (for ACL2, see, for instance [Kaufmann et al., 2000]), the above-mentioned intervention is organized by means of devising suitable *auxiliary lemmas*, or by pruning the search space.

The following generation of Theorem Provers stems from the explicit recognition of this human intervention: the *Tactical Theorem Provers* (TTPs). These computer programs allow users to direct interactively the construction of a formal proof. The most important TTPs in our context are Isabelle and Coq. Isabelle is a proof assistant for higher-order logic [Nipkow et al., 2002], designed as a theorem prover which is *generic* with respect to the logic implemented [Paulson, 1994]. Coq is an interactive proof assistant [Bertot and Castéran, 2004], based on the *calculus of inductive constructions* [Coquand and Huet, 1988].

Nowadays, each TP can be interpreted along with a continuous line ranging from pure ATP (where the input is a claim) to *Proof Checkers* (where the input is a tentative

[1]This task, in non trivial cases, is not negligible.

formal proof). At intermediate points there are systems allowing users to direct the proof construction, and, in particular, TTPs. But it is worth noting that the suggestions which can be given to the existing TPs are much closer to symbolic logic than to high level mathematical arguments (as found in any mathematical standard text). Even if some interesting theorems have been proven with these tools (for instance, the *Fundamental Theorem of Algebra* proved in Coq [Geuvers et al., 2002]) the size and complexity of these tasks illustrate convincingly the current state of TPs, with respect to the legacy mathematical background.

And thus, more research is needed in order to make TPs really usable for the standard mathematician. This memoir is devoted to such a research. In fact, our initial motivation was not of a general or philosophical nature. Our interest was based on an emerging field of application of TPs: proving the correctness of computer programs.

More specifically, our primary object of interest was a computer algebra system for Algebraic Topology calculations, called *Kenzo*. EAT [Rubio et al., 1997] and Kenzo [Dousson et al., 1999] are software systems written under Sergeraert's direction for Symbolic Computation in Algebraic Topology and Homological Algebra. These systems have produced remarkable results (for instance, some homology groups of iterated loop spaces) previously unknown. Both are based on the intensive use of functional programming techniques, which enable in particular to encode and handle at runtime the infinite data structures appearing in Algebraic Topology algorithms. As pointed out in [Calmet, 2003], Algebraic Topology is a field where challenging problems remain open for Computer Algebra Systems and Theorem Provers.

In order to increase the knowledge on the EAT and Kenzo systems, a project to formally analyze fragments of the programs was undertaken. In the recent years, several results related to the algebraic specification of data structures have been found, some of them present in [Lambán et al., 2003]. Following this process, the algorithms dealing with these data structures were our next goal, in order to give certified versions of some crucial fragments of Kenzo using the tactical theorem prover Isabelle. Some previous work in the area of Group Theory and the expressiveness of higher-order logic were the reasons to choose Isabelle for this task.

A first result for which we intend to implement a proof is the Basic Perturbation Lemma (from now on, BPL), since its proof has an associated algorithm used in Kenzo as one of the core parts of the program.

In fact, in this memoir no complete mechanized proof of the BPL is presented. During the research process, our focus moved into the study of techniques enabling systems with a higher reasoning level. Thus, we preferred to explore different approaches on a (small) fragment of the proof, instead of increasing the number of fragments proved.

Our perspective is that of an end user. That is to say, we have not considered the possibility of introducing changes in Isabelle (even if our trails could inspire some new features in the following releases of Isabelle, in particular in the mechanisms to interprete locales). Our point of view is not that of a systems engineer, but that of a (computational) mathematician. We have therefore pursued the introduction of conceptual schemes, with the aim of increasing the reasoning level in a system (Isabelle), *in its current state*. We think that some of these schemes have been found, as presented in this text.

The organization of the memoir is as follows. In Chapter 1, we introduce some preliminaries: for Mathematics (elementary Homological Algebra), TP (Isabelle) and Computer Algebra (Kenzo). Chapter 2 is devoted to an extremely detailed proof of the BPL. The level of detail is much finer than in conventional textbooks. This (a little bit) boring presentation is justified for three reasons. The first one is to make aware the reader (and, before, the author) of the large number of details that usually pass unnoticed in a single mathematical argument. The second reason is to serve as a (medium level) design of a mechanized proof. The third reason is to make clear that, even with this tedious presentation, the distance to a TP is very big. This fact will be clear when reading Chapter 3, where the actual Isabelle encoding of (fragments of) the proof starts.

Chapter 3 contains our main contributions. After introducing a test lemma, we present four approaches to mechanized reasoning in Homological Algebra. The first approach is called *symbolic* and is characterized by its high level of automation and its low expressive power. The second approach, called *set-theoretic*, tries to emulate the "theoretical" way of reasoning in standard Mathematics ("theoretical" because proofs become unfeasible very quickly due to its enormous size). The third one, called *morphism-based*, tries to use the advantages of the two previous approaches: automation and expressive power. In the fourth approach, a technical improvement recently incorporated to Isabelle (namely *locales*) is used to make our proofs shorter and more readable.

Chapter 4 briefly explores a system to link proofs and programs. The goal is not to obtain more general or efficient algorithms (that purpose would be out of the possibilities of the current technology, as it is explained in the sequel). The general aim is rather to get closer to the initial goal of our research: give certificates of correctness for Computer Algebra programs. Interestingly enough, it turned out this humble experience led us to an important foundation problem: namely, the constructive nature of our approaches, and, more generally, of Algebraic Topology itself.

The approach chosen is *code extraction*. From a formal text (a proof, a specification) an executable program is (automatically) extracted. This program is, by construction, correct with respect to the formal text used as source. In the context of TPs, the most usual approach is to extract code *from proofs* (see, for instance, [Cruz-Filipe and Spitters, 2003]). However, it turns out that Algebraic Topology statements contain, in most cases, the definition of the object to be computed, that is to say: they are *constructive statements*. Then, code can be extracted *from statements* instead of from proofs. These ideas are illustrated in Chapter 4 by means of an elementary arithmetical example (related to prime numbers). At the end of the chapter, a simple example of code extraction in the field of Computer Algebra is considered.

The memoir finishes with a chapter of conclusions and open problems and the bibliography. In Appendix A a detailed enumeration of the Isabelle files that were developed for this work can be found. The Isabelle files are online available in the web page `http://www.unirioja.es/cu/jearansa/isabellefiles/`. The total amount of Isabelle and ML code lines is ca. 8850, and the number of KB is over 500.

Chapter 1

Preliminaries

1.1 Mathematical Machinery

1.1.1 Basics on Homological Algebra

In the following definitions, some notions of Homological Algebra are briefly introduced (for details, see for instance [Mac Lane, 1994]).

Definition 1.1.1. A *graded group* C_* is a family of abelian groups indexed by the integer numbers, $C_* = \{C_n\}_{n\in\mathbb{Z}}$, with each C_n an abelian group.

Definition 1.1.2. Given A_* and B_* two graded groups, a *graded group homomorphism* $f\colon A_* \to B_*$ *of degree* k $(\in \mathbb{Z})$ is a family of group homomorphisms, $f = \{f_n\}_{n\in\mathbb{Z}}$, with $f_n\colon A_n \to B_{n+k}$ a group homomorphism for all n in $\mathbb{Z}$.

Definition 1.1.3. A *differential group* is a pair (C, d_C) where C is an abelian group and d_C is an endomorphism such that $d_C d_C = 0_{\mathrm{End}(C)}$; d_C is called the *differential map* or *boundary operator* of C.

Definition 1.1.4. Given (A, d_A) and (B, d_B) two differential groups, a *differential group homomorphism* $f\colon (A, d_A) \to (B, d_B)$ is a group homomorphism $f\colon A \to B$ which also is coherent with the differentials: $f d_A = d_B f$.

Definition 1.1.5. A *chain complex* is a pair (C_*, d_{C_*}), where C_* is a graded group, and d_{C_*} (*the differential*) is a graded group endomorphism $d_{C_*}\colon C_* \to C_*$ of degree -1 such that $d_{C_*} d_{C_*} = 0_{\mathrm{End}(C_*)}$ (in other words, for all n in $\mathbb{Z}$, $d_{C_n} d_{C_{n+1}} = 0_{\mathrm{Hom}(C_{n+1}, C_{n-1})}$).

Definition 1.1.6. A *chain complex homomorphism* $f\colon (A_*, d_{A_*}) \to (B_*, d_{B_*})$ between two chain complexes (A_*, d_{A_*}) and (B_*, d_{B_*}) is a graded group homomorphism $f\colon A_* \to B_*$ such that $f d_{A_*} = d_{B_*} f$. The degree of f will be its degree as graded group homomorphism.

Let us note that the same family of group homomorphisms $f = \{f_n\}_{n\in\mathbb{Z}}$ can be considered, depending on the source and the target, as a graded group homomorphism or as a chain complex homomorphism (clearly a chain complex homomorphism is also a graded group homomorphism).

The condition $d_{C_*}d_{C_*} = 0_{\mathrm{End}(C_*)}$ makes the following definition meaningful, since $\operatorname{im} d_{C_{n+1}} \subseteq \ker d_{C_n}$:

Definition 1.1.7. The *homology group* of a chain complex (C_*, d_{C_*}) is the family of abelian groups

$$H_n((C_*, d_{C_*})) = \ker d_{C_n} \big/ \operatorname{im} d_{C_{n+1}}, \qquad \forall n \in \mathbb{Z}.$$

The computation of homology groups of chain complexes is one of the central tasks in algorithmic Homological Algebra and Algebraic Topology. A general computing strategy consists in replacing a given chain complex by another *simpler* one, but with the same homological information. This idea can be formally expressed with the following definition:

Definition 1.1.8. Given two chain complexes (D_*, d_{D_*}) and (C_*, d_{C_*}), a *homotopy equivalence* between them is a tuple of homomorphisms (f, g, h_1, h_2) satisfying:

1. The components f and g are chain complex homomorphisms of degree 0, $f\colon (D_*, d_{D_*}) \to (C_*, d_{C_*})$ and $g\colon (C_*, d_{C_*}) \to (D_*, d_{D_*})$.
2. The component h_1 is a homotopy operator on D_*, that is to say, a graded group endomorphism of degree +1.
3. The component h_2 is a homotopy operator on C_*.
4. The following relations hold:
   - $gf + d_{D_*}h_1 + h_1 d_{D_*} = \mathrm{id}_{D_*}$;
   - $fg + d_{C_*}h_2 + h_2 d_{C_*} = \mathrm{id}_{C_*}$.

The following result shows that homotopy equivalences preserve homology groups.

Theorem 1.1.9. *Given two chain complexes (D_*, d_{D_*}) and (C_*, d_{C_*}), whenever a homotopy equivalence can be defined between them, the following relation holds:*

$$H_n(C_*, d_{C_*}) \cong H_n(D_*, d_{D_*}), \qquad \forall n \in \mathbb{Z}.$$

In our context, the most important type of homotopy equivalence is a particular case called *reduction*, introduced in Section 2.1.

Similar definitions and theorem can be applied to the ungraded case, that is to say, to differential groups.

1.1.2 Algebraic structures

In the previous section, some algebraic structures in the field of Homological Algebra have been introduced. The definitions of these algebraic structures are based on some other well-known algebraic structures. Nevertheless, we considered interesting to introduce them here, in order to maintain the similarity between the mathematical framework described here with the computer algebra environment (Kenzo) described in Section 1.2.2 and also with the theorem proving environment (Isabelle) introduced in Section 1.3.2. In those frameworks, the implementation of mathematical structures will have great influence over later developments. Another reason to introduce these definitions here is to clearly state the syntax that will be used in the sequel.

Definitions, except explicitly stated, have been extracted from [Jacobson, 1995], which is also the text used to produce the specification of these algebraic structures in the Isabelle theorem prover. Notations will be modified, trying to keep simplicity but also the resemblance with the notations introduced later for the symbolic computation system Kenzo and the theorem prover Isabelle. Definitions of algebraic structures and also of functions between them are given.

The definition of magma is based on the one by Bourbaki [Bourbaki, 1970]:

Definition 1.1.10. A *magma* is a pair (M, p) in which M is a set (or carrier) and p is a binary operation (or product) in M, verifying that it is closed for the elements of M (in symbols, $p\colon M \times M \to M$).

Definition 1.1.11. A *semigroup* is a magma (S, p) such that p is associative.

Definition 1.1.12. A *monoid* is a triple $(M, p, 1)$ such that M is a non empty set, (M, p) is a semigroup, and 1 is an element (of M) such that $p(1, a) = a = p(a, 1)$ for all a in M.

Definition 1.1.13. An element u of a monoid $(M, p, 1)$ is said to be *invertible* (or a *unit*) if there exists a v in M such that

$$p(u, v) = 1 = p(v, u)$$

Definition 1.1.14. A *group* $(G, p, 1)$ is a monoid all of whose elements are invertible (or units).

The binary operation is usually preferred in an infix style, and then the product $p(a, b)$ is denoted like $a \cdot b$, or even omitted, yielding ab.

Definition 1.1.15. A group $(G, p, 1)$ is said to be a *commutative group* if all its elements commute with respect to the binary operation, i.e., $ab = ba$ holds for all a, b in G.

In the following, when we talk about groups, we will refer to the binary operation as product, and when we talk about *abelian* groups we will call it addition. In the previous section, the definition of graded group (see Definition 1.1.15) was based on the one of abelian group.

Based on the definition of abelian group and also on that of monoids, the definition of ring can be given:

Definition 1.1.16. A *ring* is a structure $(R, +, \cdot, 0, 1)$ consisting of a non-empty set R together with two binary operations $+, \cdot,$ in R and distinguished elements $0, 1$ such that:

1. $(R, +, 0)$ is an abelian group.
2. $(R, \cdot, 1)$ is a monoid.
3. The distributive laws

$$\begin{aligned} a(b + c) &= ab + ac \\ (b + c)a &= ba + bc \end{aligned}$$

   hold for all a, b, c in R.

The structure $(R, \cdot, 1)$ is called the multiplicative monoid of $(R, +, \cdot, 0, 1)$ and $(R, +, 0)$ is called the additive (abelian) group of the ring.

Substructures of algebraic structures will be also relevant for our work. Following the definitions in [Jacobson, 1995], we will define them in terms of *subsets* of the carrier set of the algebraic structure.

Definition 1.1.17. Given a magma (M, p), a subset N is said to be a *submagma* if for all a, b in N, ab is also in N (i.e., if it is closed under the binary operation).

Definitions of submonoid and subgroup are then given in terms of submagmas.

Definition 1.1.18. Given a monoid $(M, p, 1)$, a submagma N of (M, p) is said to be a *submonoid* if it contains the element 1.

Definition 1.1.19. Given a group $(G, p, 1)$, a submagma M of $(G, p, 1)$ is said to be a *subgroup* if it verifies that 1 belongs to M, and for all m in G, the inverse of m is also an element of G.

Homomorphisms represent transformations between algebraic structures. They are useful to transfer properties between the algebraic structures, and therefore they are crucial in the field of Universal Algebra. They will be also relevant for our work. They have already appeared in Section 1.1.1, in the form of differentials of chain complexes and homomorphisms between them.

Definition 1.1.20. Given two magmas (M, p) and (M', p'), a map f of (M, p) into (M', p') is called a *homomorphism* if

$$f(p(a, b)) = p'(fa, fb) \qquad \forall a, b \in M.$$

In the case where $(M, p) = (M', p')$, f is said to be an *endomorphism*.

The set of homomorphisms between two magmas (M, p) and (M', p') will be denoted by $\text{Hom}(M, M')$. Only if confusion can arise, the set will be denoted, in a more verbose version, as $\text{Hom}((M, p), (M', p'))$.

The set of endomorphisms of a magma (M, p) will be denoted by $\text{End}(M)$.

The given definition of homomorphism between magmas is also valid for semigroups, monoids and groups, with the corresponding modifications.

Definition 1.1.21. Let f be an endomorphism of a magma (M, p), and n in $\mathbb{N}$. The *n-th power* of f will be denoted as f^n, and it is defined, for all x in M, as $f^n(x) = x$ if $n = 0$, and $f^n(x) = f(f^{n-1}(x))$ if $n > 0$.

Due to the importance of homomorphisms, some special sets related to them are also defined:

Definition 1.1.22. Given a homomorphism f between two magmas (M, p) and (M', p'), the image set of f is defined as the following subset of M':

$$\operatorname{im} f \equiv \{f(x) \mid x \in M\}$$

Definition 1.1.23. Given a homomorphism f between two monoids $(M, p, 1)$ and $(M', p', 1')$, the kernel of f is defined as

$$\ker f \equiv \{x \in M \mid f(x) = 1'\}$$

Finally, the ringoid (or preadditive category) algebraic structure, usually found in Category Theory, is also introduced. We will see that it fits quite precisely to represent some of the mathematical settings we will deal with in the memoir. The definition is extracted from [Mac Lane, 1994]:

Definition 1.1.24. A *ringoid* R is a category whose morphism sets $\text{Hom}(X, Y)$ are abelian groups in such a way that composition is bilinear. The endomorphism set $\text{End}(X) = \text{Hom}(X, X)$ of an object X has a ring structure with product given by composition of morphisms. Conversely any ring R can be identified with the ringoid with a single object whose endomorphism set is R.

1.2 Computing Machinery

1.2.1 Basics on the Kenzo system

The following presentation of the Kenzo system has been essentially extracted from [Rubio and Sergeraert, 2002, Sections 7 and 8]:

Kenzo is a 16000 lines program written in Common Lisp ([Graham, 1996, Steele, 1990]), www-available and with a rich documentation (see [Dousson et al., 1999] for documentation and details). It can be used with any Common Lisp system satisfying the ANSI norm. Its original purpose was the computation of homology and homotopy groups in Algebraic Topology. The main tool used in Kenzo to solve this problem is functional programming.

By using functional programming, some techniques in Algebraic Topology are encoded in the form of algorithms. Moreover, the possibility to implement in a computer some infinitely generated objects and to do computations with them is obtained. The results obtained by Kenzo in the computation of homology groups are relevant, and even some of them have not been reached by any other means. The predecessor of Kenzo, called EAT (Effective Algebraic Topology, see [Rubio et al., 1997, Rubio et al., 1998]), was mainly devoted to the homology of iterated loop spaces. Kenzo, in addition to this, improves the performance and increases the range of objects which can be dealt with. Some interesting examples that have been computed with the help of Kenzo are $H_5\Omega^3 \mathit{Moore}(\mathbb{Z}_2, 4)$, the fifth homology group of the three-iterated loop space of $\mathit{Moore}(\mathbb{Z}_2, 4)$. Also homotopy groups can be computed; the computation of $\pi_5(\Omega S^3 \cup_2 e^3)$ is an example of it (a 3-cell is attached to a loop space ΩS^3 by a map of degree 2, and the fifth homotopy group is then computed).

These processes of computation require a great amount of algebraic structures and equivalences to be built, and thus a lot of resources. The reliability of this software is out of question, but nevertheless, the algorithms associated to these computations are of interest in themselves.

1.2.2 Data structures in Kenzo

As it has been remarked in the previous section, Kenzo is a system for symbolic computation in Homological Algebra. Therefore, it is able to compute with Homological Algebra structures such as the ones introduced in Section 1.1.1, and even with more complicated ones. Here we give an overview of how these structures are represented in the Kenzo system, using object oriented and functional programming features simultaneously.

Following the bottom-up presentation already used in Sections 1.1.1 and 1.1.2, we start by introducing the implementation of a very basic algebraic structure in Common Lisp (this is closer to the style of EAT [Rubio et al., 1997, Rubio et al., 1998], the predecessor of Kenzo, than to that of Kenzo itself). In Definition 1.1.11, a semigroup (S, p)

was said to be a set with a binary operation over it, which satisfies associativity. The Common Lisp implementation of this structure could be as follows:

```
(defstruct semigroup
    carrier
    product
    equality
    )
```

As it can be seen, a record is defined with three fields which represent the carrier of the algebraic structure, a comparison test or `equality` between elements of the `carrier`, and also a product. The `carrier` field refers to a set of Common Lisp data *representing* the elements of the set S. "Representing" does not mean to be *equal* to the elements of the set S. The reason is that an interpretation of the data is needed (see the definition of *abstraction function* in Section 3.4).

In particular, the carrier can denote some representatives of an equivalence relation, defined by the field `equality`, which is a boolean function taking two arguments (defining an equivalence relation). From an operational point of view, the `equality` is also necessary in order to compare elements acting as generators of free structures (the paradigmatic structure is the *free* chain complex associated to a topological space; see [May, 1967] for the definition and bellow for details on the Kenzo implementation). In particular, a `carrier` could denote the set of the integer numbers $\mathbb{Z}$, and the equality would allow the system to work *modulo 3*, thus faithfully representing the finite set $\mathbb{Z}/3\mathbb{Z}$.

As the example shows, and is frequently mandatory in Algebraic Topology, infinite sets appear in Kenzo. Three possibilities are then available:

1. The field `carrier` could contain a list of elements (the set S being then necessarily finite).
2. The unary predicate (boolean function) could act as a test function over a set (the finite of infinite nature of S depends then on the `equality` function; infinite sets are allowed).
3. A label `:locally-effective` could indicate that no information about the finite or infinite nature of S is known or available.

Therefore, the set S is more accurately represented by the couple `<carrier, equality>`. Starting from this representation of sets, the hierarchy of algebraic structures (magmas, semigroups, monoids, ...) could be built (by using the Common Lisp extension tool `:include` of `defstruct`, or, more properly, using the object-oriented features available in Common Lisp), following the representation given in Section 1.1.2. Instead of that, we have presented the semigroup structure as a whole. The last field to be described is `product`, which contains a binary function closed over the elements of the carrier, and coherent with respect to the `equality`.

The most important difference between the mathematical definition and its implementation is that no axiomatic information appears at all. Kenzo, being a system for computing, does not need any information about the properties of the operations, only their behaviour is relevant. As a consequence of this, semigroups can be implemented in Common Lisp similarly to magmas, even when their mathematical definitions differ.

Making use of the existing tools in Common Lisp for inheritance between classes (in CLOS, the Common Lisp Object System), more elaborate structures and relations among them are represented in Kenzo. For instance, the following class definition corresponds to the Kenzo definition of chain complex:

```
(DEFCLASS CHAIN-COMPLEX ()
    ((cmpr :type cmprf :initarg :cmpr :reader cmpr1)
     (basis :type basis :initarg :basis :reader basis1)
     ;; BaSe GeNerator
     (bsgn :type gnrt :initarg :bsgn :reader bsgn)
     ;; DiFFeRential
     (dffr :type morphism :initarg :dffr :reader dffr1)
     ;; GRound MoDule
     (grmd :type chain-complex :initarg :grmd :reader grmd)
     ;; EFfective HoMology
     (efhm :type homotopy-equivalence :initarg :efhm :reader efhm)
     ;; IDentification NuMber
     (idnm :type fixnum :initform (incf *idnm-counter*) :reader idnm)
     ;; ORiGiN
     (orgn :type list :initarg :orgn :reader orgn)))
```

The relevant fields are `cmpr`, a function coding the equality between the generators (recall that a chain complex in Kenzo is *free*, and so generated by a set); `basis`, which stores information about the generators of the chain complex, in the form of a function or also as the label `:locally-effective`; `dffr` represents the differential of the chain complex, which, as stated in Definition 1.1.5, will be a graded group homomorphism (and thus, its type in Kenzo is `morphism`; see below). The field `efhm` allows the system to store information about homotopy equivalences, as presented in Definition 1.1.8, making possible the relationship with a chain complex whose homology is directly computable. The remaining fields are used to keep record of information about the inner organization of the system, and will not be relevant for us.

Some facts can be observed from the previous definitions. First, the use of functional programming, as far as some fields of these structures have functional nature, such as `cmpr` or `basis`, in the form of primitive functions or lexical closures. Second, the very different nature of the algebraic structures is coded following a similar pattern, allowing the system to speed up numerous computations. Consequently, the field representing the basis of the chain complex receives quite different values. Its value depends on the nature of the algebraic structure we are dealing with. In cases where this algebraic structure is finite, or *effective* in Sergeraert's terminology, the field value will be a function assigning to each dimension a list with the elements of the basis of the chain complex in such dimension. Not every chain complex is effective, and sometimes infinite basis can appear in some dimensions. Nevertheless, computations can be carried out also with these structures. These structures are known as *locally effective* in Sergeraert's terminology, and the value of its `basis` field is equal to `:locally-effective`.

Chain complexes are the most simple algebraic structure implemented in Kenzo. From them, by inheritance, are defined the rest of algebraic structures. From a practical point of view, the class `CHAIN-COMPLEX` is extended by inheritance with new fields, obtaining more elaborate structures. For instance, extending with a `cprd` field, we obtain the `COALGEBRA` class from the `CHAIN-COMPLEX` class. Multiple inheritance is also available in CLOS; for instance, the class `SIMPLICIAL-GROUP` is obtained by inheritance from classes `KAN` and `HOPF-ALGEBRA`.

Homomorphisms between algebraic structures, as introduced in Definition 1.1.6, are one of the most used tools in Kenzo, since they allow the system to establish homotopy equivalences, and thus, to compute the homology of some locally effective algebraic structures in terms of the homology of some effective algebraic structures. In Kenzo they are also defined as a class with name `MORPHISM`, which definition is as follows:

```
(DEFCLASS MORPHISM ()
     ;; SOuRCe
    ((sorc :type chain-complex :initarg :sorc :reader sorc)
     ;; TaRGeT
     (trgt :type chain-complex :initarg :trgt :reader trgt)
     ;; DEGRee
     (degr :type fixnum :initarg :degr :reader degr)
     ;; INTeRnal
     (intr :type intr-mrph :initarg :intr :reader intr)
     ;; STRaTegy
     (strt :type strt :initarg :strt :reader strt)
     ;; CaLl NuMber
     (???-clnm :type fixnum :initform 0 :accessor ???-clnm)
     (?-clnm :type fixnum :initform 0 :accessor ?-clnm)
     ;; ReSuLTS
     (rslts :type simple-vector :reader rslts)
     ;; IDentification NuMber
     (idnm :type fixnum :initform (incf *idnm-counter*) :reader idnm)
     ;; ORiGiN
     (orgn :type list :initarg :orgn :reader orgn)))
```

The source and target of homomorphisms are explicitly stored in the fields `sorc` and `trgt`. In the field `degr`, the degree of the homomorphism is stored (for instance, differentials have degree -1). The field `intr` contains the algorithm representing the homomorphism behaviour, and works in combination with the field `strt`, which determines how the `intr` field must be applied to its arguments. The other fields are of internal use for the system. For instance, the field `rslts` provides information about results already computed to avoid repeating computations.

Kenzo is designed for *computing* with algebraic structures (and not for *proving* facts with them). This influences the design of classes in the Kenzo system. In Section 1.3.2 we will introduce the Isabelle representation of algebraic structures, and in Sections 3.5.2, 3.6.3, 3.7.3, and 3.8.2, different representations of homomorphisms will be considered also in this sytem. There we will comment on the differences and similarities between their representation in Kenzo and the ones introduced in Isabelle.

1.3 Deduction Machinery

1.3.1 Basics on Isabelle

Information in this section has been mainly extracted from [Paulson, 1989, Paulson, 1990a, Paulson, 1990b, Paulson, 1992, Nipkow et al., 2000, Nipkow et al., 2002]. Isabelle is a generic theorem prover which historical origins are the family of provers LCF (Logic of Computable Functions). These family of provers sought a compromise between fully automatic theorem proving and simple step proof checking. The LCF provers are usually developed in a meta-language which allows the user to give a representation of the logic. The idea behind these provers is that terms and formulas are computable data; then, proofs can be built in a forward style, from the premises to the goal, and so inference rules are functions from theorems to theorems. Proofs can be also built in a backwards style, starting from the goal. An LCF tactic is then a function that reduces a goal to zero or more subgoals. When all the subgoals have been solved, LCF can recover the proof in a forward style, yielding the desired theorem.

Isabelle was developed by L. C. Paulson using the functional programming language ML (more concretely, Standard ML) in an attempt to show that it was a practical alternative to Lisp. At the same time, new ideas and some experimental processes (for instance, implementation of higher-order unification) were introduced to overcome some of the difficulties found in the then existing LCF-style systems, mainly HOL [Gordon and Melham, 1993], Nuprl [Constable et al., 1986] and the pioneer Edinburgh LCF [Gordon et al., 1979].

Even more ambitious, the intention was to build a generic theorem prover. To be precise, the system should be based on a small set of rules, forming the meta logic of the system (Isabelle/Pure), and new logics should be implemented on top of it. At present, it provides implementations of, at least, various first-order logics, Zermelo-Fraenkel set theory, higher-order logic and Scott's logic of computable functions. The logic implemented in Pure consists in a fragment of higher-order logic, with typed λ-calculus, together with implication ($\Longrightarrow$), universal quantification ($\bigwedge$) and equality ($\equiv$). A set of rules are also provided representing introduction and elimination of the previous connectors.

Now, new object-logics can be implemented making use of the existing rules. The elements that have to be defined to implement a new logic are types, constants and axioms. From a syntactical point of view, the elements of the object-logics will be mapped to elements of the meta-logic.

Our interest will be focused on the implementation of higher-order logic (HOL) present in the standard distribution of Isabelle. As it is said in the Isabelle web page, "Isabelle/HOL is currently the best developed object logic, including an extensive library of (concrete) mathematics, and various packages for advanced definitional concepts like (co-)inductive sets and types, well-founded recursion, etc". This implementation is based on the HOL system by Gordon (see [Gordon and Melham, 1993]), which itself tries to implement the ideas given in Church's paper [Church, 1940]. More concretely, a simple type theory is given, where function and product types are defined (as well as a *bool* type for formulas). Then, logical connectives are defined; first, implication ($\longrightarrow$) and quantification ($\forall$) (based on the definitions given in the meta-logic Pure), and then the rest of logical constants and connectives, such as equality ($=$), True, False, negation, conjunction, disjunction, existence, and so on, are defined in terms of the basic ones. A concise description from the theoretical point of view of the HOL definition is given, for instance, in [Paulson, 1990a]; details about implementation issues in Isabelle are exposed in [Nipkow et al., 2000].

Isabelle/HOL has already shown its effectiveness for implementing some (non-trivial) large proofs in mathematics. One of these examples can be found in [Kammüller and Paulson, 1999], where a complete proof of the Sylow's Theorem is produced, making use of both arithmetic and abstract algebra concepts; another large development in functional analysis can be found in [Bauer and Wenzel, 2000], where a proof of the Hahn-Banach Theorem is implemented; one of the last examples of such developments is a formalization of the prime number theorem (details can be found in [Avigad, 2004]).

These examples, as well as some others in different theorem proving tools, have produced two effects. First, they have aroused an increasing interest from mathematicians in the capabilities of mechanized reasoning. Second, a fast development of these tools has been produced. In the case of Isabelle, its first versions were designed for backward proving, starting from a goal and splitting it up into subgoals, that finally are matched with the premises. Proofs were not very readable and far from the "human style". The Isar [Wenzel, 2002] extension to Isabelle eases the implementation of proofs in a readable way, and not only in a backward style, but also building the goal in a forward direction, starting from the premises. In addition to this, it provides the user with tools to translate the Isabelle files into LaTeX files, allowing the user to obtain files where the lemmas and their proofs are nicely displayed. The features offered by Isar, also for obtaining LaTeX versions of lemmas and proofs, are used in this memoir.

1.3.2 Structures in Isabelle

In this section, an introduction to the implementation of algebraic structures in Isabelle is given. They will be intensively used in Section 3.5 and onwards. Some other necessary concepts about Isabelle syntax will be also introduced for a better understanding of this memoir.

In previous Sections 1.1.2 and 1.2.2, the importance of algebraic structures when trying to define a mathematical setting has been pointed out. Their implementation in theorem proving environments is a well-known problem. Most of the theorem proving systems offer an implementation of modules that can be used to this aim. In Isabelle/HOL the implementation of algebraic structures can be done in various ways. Axiomatic classes [Wenzel, 2005] can be the first option considered, due to its simplicity and the easy way of working with them. Starting from a collection of axioms, proofs in the created environment can be developed. The main disadvantage is that axiomatic classes are not first class citizens of the system, and thus, working with them in general is limited. A second option, and the one preferred here, is the use of record types for encoding the operators of the algebraic structure; in addition, one must specify the rules satisfied by the algebraic structure. Records can be implemented in the type system of Isabelle/HOL, and therefore reasoning can be done with the operators of the record structure, but also in a generic way, allowing the user to state general properties about the record structures. Two reasons made us choose the second option. First, axiomatic classes are restricted to a single carrier structure, which means that we could consider endomorphisms over a given group as an axiomatic class, but not, for instance, homomorphisms in general between two groups. Our theorems deal with various algebraic structures, their endomorphisms and homomorphisms among them, and we considered record types more appropriate. A second reason can be also pointed out. The implementation of algebraic structures in Kenzo, as exposed in Section 1.2.2, is done through classes with fields representing the operators of the algebraic structure. In the Isabelle/HOL type system there are no object-oriented features, but nevertheless, record types resemble the notion of classes, and instances of the record types the one of objects, and thus records seem to be the best way to represent algebraic structures in order to maintain the similarity with Kenzo.

Here we follow the model of implementation of algebraic structures given in [Kammüller, 1999], and part of the notation and some conventions have been also extracted from [Loeckx et al., 1996].

In Isabelle/HOL definition of sets over a generic type is available, and therefore, the elements of a generic algebraic structure will be represented as elements of a type $'a\ set$. Here $'a$ represents a generic type in the Isabelle/HOL type system, that can be later instantiated to more complex types, resembling the variable types in ML, whereas *set* is a type constructor. Sets in Isabelle are simply predicates over types. The definition of a set involves two steps; first it is declared of a certain type, and then its definition is given through the collection of properties satisfied by the elements of the set. The main operation then over sets is "membership", whose type definition in the Isabelle/HOL library is:

```
consts
  op :    :: 'a => 'a set  => bool     (( - / :  - ) [ 50, 51 ] 50 )    --    membership
syntax (xsymbols)
  op :    :: 'a => 'a set  => bool     (( - / ∈ - ) [ 50, 51 ] 50 )
```

Let us observe that, in its type definition, the operation is polymorphic (since $'a$ is a generic type), and for any object x and a set A of the same type as x, the expression $x\colon A$ will be of type *bool* (depending on the premises, sometimes it could be proved to be *False* or *True*). A syntactic equivalent for this operator will be the operator $\in$, that we also use in the sequel. Both operators are defined to be infix, and their precedence is 50 (in the Isabelle range, varying from 0, the highest, to 1000).

The equivalence between sets and predicates in the Isabelle/HOL logic is established through the following axioms extracted from the Isabelle *Set* theory (in Isabelle notation, braces are used to denote sets):

axioms
mem-Collect-eq [*iff*]: $(a : \{x.\ P(x)\}) = P(a)$
Collect-mem-eq [*simp*]: $\{x.\ x{:}A\} = A$

The carrier of an algebraic structure is then represented through a term of type $'a\,set$ (more complex types can be used if needed; if we are implementing an algebraic structure product of two algebraic structures, its type would be $('a \times' b)\,set$). The operations of the algebraic structure will be defined over this set. Here, a first difference with the Kenzo implementation is observed, due to the different type systems of Isabelle/HOL and Common Lisp. There, no type assignment was made for the carrier set of the algebraic structures. The generic type $'a\,set$ plays in Isabelle the role of the pair `<carrier, equality>` in Common Lisp, mentioned in Section 1.2.2.

An algebraic structure can be seen as a set of mathematical objects. First, we are going to specify algebraic structures as abstract data types, and then a possible implementation of these abstract data types in Isabelle will be introduced. Following the definitions in [Loeckx et al., 1996], a loose specification is a pair (Σ, Φ), where Σ is a signature and Φ a set of formulas. We can specify an abstract data type through its loose specification (actually, loose specifications are quite similar to the way algebraic structures are defined in Isabelle, since both definitions require a signature and a collection of axioms). The signature $\Sigma = (S, \Omega)$ will consist in the set of sorts S over which operations are defined, and Ω will be the set of operations with their arities. A syntactical description of a data type D by its operations and formulas can be given as

$$\begin{array}{l} \textbf{signature}\quad D \\ \quad x_1 \in A_1 \\ \quad \ldots \\ \quad x_n \in A_n \\ \quad P_1 \\ \quad \ldots \\ \quad P_m \end{array}$$

where A_i represent the arities of the operations, with $i \in \{1, \ldots, n\}$. The P_j are the formulas in which the operations x_i can appear, with $j \in \{1, \ldots, m\}$.

Now we can associate to D the set of all models that satisfy both the arities and also the formulas of the loose specification.

$$[\![D]\!] \equiv \{(x_1, \ldots, x_n) \in A_1 \times \ldots \times A_n \mid P_1 \wedge \ldots \wedge P_m\}$$

where in P_i any of $\{x_1, \dots, x_n\}$ can occur. This set of models is an abstract data type.

Data types are divided into *simple data types* or *parameterized data types* depending on whether they are parameterized by other structures or not.

The data types we are dealing with are simple, and therefore we do not present here the parameterized case (it can be found in [Kammüller, 1999]). Simple structures will then be represented as predicates over records. Record types are used as a template for the set of operations Ω of the signature. They are provided with selectors, which are projection functions enabling the reference to the constituents of a simple structure.

Records are not a primitive type in the HOL type system, but implemented through tuples. The main difference to tuples is that the elements of a record are labeled with the selectors which allow us to directly access to each component. Extensible records are a generalization of records. They will be relevant for our work because they enhance record types with parametric polymorphism and structural subtyping; in other words, they model an inheritance mechanism among record types. They were first introduced in Isabelle in [Naraschewski and Wenzel, 1998]. A *scheme* offers a way to refer to a family of records. For instance, the scheme $\{x = a, y = b, \dots\}$ represents the family of records with a field named x which value is a, and another field y which value is b. Thus, both records $\{x = a, y = b\}$ and $\{x = a, y = b, z = c\}$ are instances of the previous scheme. The same idea can be applied to types. We can define a *type scheme* as a record type which enables to represent a family of types (and thus, providing the type system with structural subtyping). The record type scheme $\{x :: A, y :: B, \dots\}$ represents, for instance, the record types $\{x :: A, y :: B\}$ and $\{x :: A, y :: B, z :: C\}$, offering a way to get polymorphism. From this representation it can be inferred that multiple inheritance will not be available, pointing out a new difference with the CLOS type system. Now the use of records to implement signatures of simple data types will be illustrated, and later the importance of extensible records to implement algebraic structures making use of inheritance will be shown.

For the tuple of operations $par \equiv (x_1, \dots, x_n)$ of a simple data type we define an Isabelle record type $'a\ par\text{-}sig$ as

$$\begin{array}{ll} \textbf{record}\ 'a\ par\text{-}sig & \equiv \\ \_.\langle x_1 \rangle :: A_1 & \\ \dots \quad\ \ \dots & \\ \_.\langle x_n \rangle :: A_n & \end{array}$$

The underscore defines the arguments position for the field selectors of this record. For example, if T is a term of appropriate record type, i.e., a suitable $n-$tuple, we can select the field x_j of T by $T.\langle x_j \rangle$.

The list of formulas Φ characterizing an abstract data type is also given in terms of axioms expressed in the Isabelle logic, which in our case is the implementation of higher-order logic, HOL. These axioms will be the ones defining the predicate which distinguish between the objects of the specified type which satisfy the given specification and the ones which do not satisfy it.

The representation of the models of a simple structure is given in the form of a predicate over records, which due to the equivalence between predicates and sets, can be also seen, with minor changes, as a set of records[1]; the record type defines the pattern of the elements of the structure. An instance of a class in the Kenzo system would be then identified with a distinguished element of this set of records.

For instance, the specification of a monoid G can be seen as a carrier set, $'a\,set$, together with a binary operation, mult, on this set, with the binary operation satisfying associativity, and with a distinguished element, one, such that it is a unit for the binary operation. With these requirements, the syntactical description of a monoid can be defined as follows:

$$\begin{array}{ll}
\textbf{signature} & \textit{monoid} \\
\quad \mathsf{mult} \in & \mathsf{carrier} \times \mathsf{carrier} \to \mathsf{carrier} \\
\quad \mathsf{one} \in & \mathsf{carrier} \\
\quad \forall x \in \mathsf{carrier}\,. & x\,\mathsf{mult}\,\mathsf{one} = x \\
\quad \forall x, y, z \in \mathsf{carrier}\,. & \!\!\!\! x\,\mathsf{mult}(y\,\mathsf{mult}\,z) = (x\,\mathsf{mult}\,y)\,\mathsf{mult}\,z
\end{array}$$

Therefore, the set of models associated to this example is formed by the elements with the given arity that also satisfy the list of axioms:

$$\begin{aligned}
[\![\,\textit{monoid}\,]\!] \equiv \{(\mathsf{carrier}, \mathsf{mult}, \mathsf{one}) \in (\mathsf{carrier} \to \mathsf{carrier} \to \mathsf{carrier}) \times (\mathsf{carrier}) \mid \\
(\forall x, y, z \in \mathsf{carrier}\,.\,x\,\mathsf{mult}(y\,\mathsf{mult}\,z) = (x\,\mathsf{mult}\,y)\,\mathsf{mult}\,z) \\
\wedge\ (\forall x \in \mathsf{carrier}\,.\,\mathsf{one}\,\mathsf{mult}\,x = x) \\
\wedge\ (\forall x \in \mathsf{carrier}\,.\,x\,\mathsf{mult}\,\mathsf{one} = x)\}
\end{aligned}$$

The implementation of algebraic structures in Isabelle can be done in a similar way as it has been done for abstract data types. First, they have to be assigned a type corresponding to the signature Σ. The set of formulas Φ will be identified with a collection of axioms in Isabelle/HOL logic. The similarity between the specification and the Isabelle implementation will be observed in the case of monoids, whose specification has been already given. Not every algebraic structure will be used on every approach (for instance, in the symbolic approach exposed in Section 3.5, just the ring structure will be used), but due to the close relationship among their definitions we have considered more appropriate to present them together. Most of these definitions are extracted from the Isabelle libraries dealing with Algebra, more concretely, they have been extracted from theories *Group* and *CRing* developed by Clemens Ballarin (see [Isa, 2005]).

Definitions are based on the ones found in [Jacobson, 1995] and here in Section 1.1.2, showing the similarities between mathematical definitions and the corresponding ones in Isabelle/HOL.

[1]Actually, in previous releases of Isabelle, algebraic structures were represented through sets of records.

record *'a partial-object* =
  carrier :: *'a set*

First, a partial object type is declared as a record with one field, a carrier set, of a generic type *'a*. Let us observe that no operation has been declared for this object. This partial object enables the definition of a membership predicate for the elements of algebraic structures (see again the analogy with the pair `<carrier, equality>` in Common Lisp definitions, presented in Section 1.2.2). Not every term of the specified type *'a* has to be member of the algebraic structure. Therefore, algebraic structures can be considered partial over their types. The consequences of this implementation when dealing with functions between algebraic structures will be explained later in Section 3.6.3. From this record type, by adding (making use of extensible records) a binary operation, a semigroup record type can be defined:

record *'a semigroup* = *'a partial-object* +
  mult :: [*'a*, *'a*] => *'a* (**infixl** $\otimes_1$ *70*)

The defined type $['a,'a] =>' a$ is short for $'a =>' a =>' a$ (and will be used indistinctly in the sequel). The right hand side of every field of the record can include additional information about infix notation, pretty print notation, or precedence related to other operations, which will be valid inside of the *locales* environment. In the case of the *mult* field of the record, the annotation declares the infix $\otimes$ operator to be a special symbol for the *mult* operator, and nested to the left. Then, if we are inside of a locale where G is a fixed term declared to be a semigroup, the expression $a \otimes b \otimes c$ stands for *mult G (mult G a b) c*. A subindex allows the operator $\otimes$ to appear indexed when we are in a locale context where more than one structure has been fixed.

The binary operation can be defined to be closed over the elements of the carrier, obtaining then a magma[2]:

locale *magma* = *struct G* +
  assumes *m-closed* [*intro*, *simp*]:
    [| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $x \otimes y \in$ *carrier G*

In Isabelle notation, $[|R_1; \ldots ; R_n|]$ is abbreviate for nested implication. Thanks to the properties of locales, the *semigroup* record type scheme is directly inferred for the parameter G in the *magma* definition. More information on locales can be found in [Ballarin, 2004]. Here, we will restrict ourselves to comment only on the features needed. Locales provide support for modular reasoning in Isabelle. As far as they are named (e.g, *magma* in the previous case), they are persistent. Facts proved in a locale context can be exported and used out of the locale. By now, it is enough to know that the previous locale definition automatically introduces a couple of lemmas called

[2]In the sequel, pretty printing symbols as well as infix notation will be used.

magma-def and *magma-axioms-def*, allowing the user to unfold the list of axioms of the magma definition. Furthermore, the locale definition provides us with a predicate (called *magma* in this case) that allows the system to determine the objects having the right type and satisfying the collection of axioms (in terms of abstract data types, this predicate allows us to determine the models of the given specification).

Now, by adding associativity of *mult* to the already specified magma, we obtain a semigroup (which is also assigned a semigroup record type, as far as it has, at least, a *mult* operation).

locale *semigroup* = *magma* +
  assumes *m-assoc*:
    [| $x \in$ *carrier G*; $y \in$ *carrier G*; $z \in$ *carrier G* |] ==> $(x \otimes y) \otimes z = x \otimes (y \otimes z)$

Locales also allow us to invoke other locales, or in other words, to import in the defined local context the conditions established in previous contexts. In the case of semigroups, this feature is applied to the axioms, which are added to the ones stated for magmas.

From the semigroup type definition, the *monoid* record type is defined:

record *'a monoid* = *'a semigroup* +
  one :: *'a* ($\mathbf{1}_1$)

Now the similarities between the record definition with the given signature of the monoid specification can be observed. Each operation of the signature can be identified with one field of this record with identical arities. We have taken advantage again of extensible records. The semigroup record type gives place to a scheme record type, that can be later extended with new fields. The record type *monoid* is the same that will be later used for groups.

The list of axioms given for the Isabelle characterization of monoids can be compared with the list of formulas Φ given in the specification of monoids previously introduced:

locale *monoid* = *semigroup* +
  assumes *one-closed* [*intro*, *simp*]: $\mathbf{1} \in$ *carrier G*
    and *l-one* [*simp*]: $x \in$ *carrier G* ==> $\mathbf{1} \otimes x = x$
    and *r-one* [*simp*]: $x \in$ *carrier G* ==> $x \otimes \mathbf{1} = x$

Groups will be also assigned a monoid record type. In order to give an axiomatic characterization of groups, first the set of *Units* of a monoid[3] is defined:

constdefs

[3]Note the occurrence of the existential quantifier $\exists$, which could place ourselves outside of constructive logic.

```
Units :: ('a, 'm) monoid-scheme => 'a set
Units G == {y. y ∈ carrier G &
                (∃ x ∈ carrier G. mult G x y = one G & mult G y x = one G)}
```

Then, a group will be a monoid whose elements are all units, or following the definition of *Units*, whose elements are all invertible:

```
locale group = monoid +
  assumes Units: carrier G ⊆ Units G
```

Abelian groups are also assigned a *monoid* record type. An abelian group is defined as:

```
locale abelian-group = abelian-monoid +
assumes a-comm-group: comm-group (| carrier = carrier G, mult = add G, one = zero G |)
```

where the only axiom added to obtain commutative groups from groups is

```
m-comm: [| x ∈ carrier G; y ∈ carrier G |] ==> x ⊗ y = y ⊗ x
```

The Isabelle definition of differential group (see Definition 1.1.3) was introduced in our work. Its type definition is inherited from the one of monoid. A new field is added representing the differential of the differential group:

```
record 'a diff-group = 'a monoid +
  diff :: 'a => 'a (differı 81)
```

and it satisfies the following axioms, including the ones of abelian groups:

```
locale diff-group = abelian-group CC +
  assumes diff-hom : diff CC ∈ hom CC CC
  and  diff-nilpot : ∀x. x ∈ carrier CC ==> ( diff CC (diff CC x) = 0 )
```

The differential must be an endomorphism of the abelian group satisfying that composed with itself is null. As in the data structure for chain complexes used in Kenzo (see Section 1.2.2), the definition of a differential group in Isabelle makes reference to homomorphisms (since the differential must be such a structure).

The only algebraic structures besides the enumerated ones that will be used are rings. Rings are not included in Isabelle in the same library as groups. However, their implementation is based on the algebraic structures appearing there. The record type declared for rings is obtained by extending the record type for monoids with a new constant *zero* and a binary operation *add*:

```
record 'a ring = 'a monoid +
  zero :: 'a (0₁)
  add :: ['a, 'a] => 'a (infixl ⊕₁ 65)
```

From the ring type definition, where an underlying monoid and an abelian group can be found, the Isabelle type for rings could be thought of as a record inheriting from the monoid and the abelian group types. This approach is not valid for the ring type definition, since multiple inheritance is not possible in the Isabelle type system by using extensible records; nevertheless, it is valid for the axiomatic definition. The ring satisfies with respect to the multiplicative operation and the constant *one* the axioms of a monoid, and with respect to the additive operation and the constant *zero* the axioms of an abelian group. In addition to this, distributivity of the multiplicative operation with respect to the additive one has to be satisfied. The collection of axioms which defines a ring is given then by:

```
locale ring = abelian-group R + monoid R +
  assumes l-distr: [| x ∈ carrier R; y ∈ carrier R; z ∈ carrier R |]
      ==> (x ⊕ y) ⊗ z = x ⊗ z ⊕ y ⊗ z
   and r-distr: [| x ∈ carrier R; y ∈ carrier R; z ∈ carrier R |]
      ==> z ⊗ (x ⊕ y) = z ⊗ x ⊕ z ⊗ y
```

In the previous code can be observed the advantage of working with parametric polymorphism, which has permitted us to state the property *monoid R*, even when *R* has a record type *ring* which is an extension of (but not equal to) the *monoid* record type. The predicate *monoid* is defined for the record type *monoid* and for every record type obtained from this one by extension.

A relevant feature of the implementation proposed for algebraic structures, already pointed out, is that they have a generic type. The Isabelle type *'a* represents a variable type that can be instantiated with more complex types (such as functional ones, for instance). Taking advantage of this feature, the previous definition of ring will be used in Sections 3.7 and 3.8 to implement rings of endomorphisms, with the endomorphisms defined as objects with a record type, in the first case, and with a functional type in the second one. In a more simple example, we can define a record with a carrier (for instance, the generic set of type *'a* can be instantiated with the set of all the integers) and the required operations (following with the example, 1, and usual product of integers). The record in the example can be proved to satisfy the monoid axioms (in a quite straightforward way):

```
lemma shows  monoid (| carrier = UNIV , mult = op * , one = (1::int) |)
  by (unfold monoid-def monoid-axioms-def semigroup-axioms-def magma-def) (simp)
```

Once we have proved this record to satisfy the axioms of some algebraic structure of the Isabelle library, we will have the chance to apply every proved lemma for this algebraic structure to the structure we have defined. The set of all integers, with the suitable operations, can be also proved to be a ring, but even this proof, where several basic facts about the integers are available in the Isabelle library, requires a greater proving effort than the previous one about the monoid. This fact could be illustrative of the complexity of such a proof when we choose as carrier set more complicated terms, such as functional or record ones.

Some other Isabelle features and properties will be relevant for our work. For instance, representation of functions and the principle of extensionality, as implemented in Isabelle, will greatly influence on the representation of homomorphisms. We considered more appropriate to introduce these questions in Chapter 3; first, in order to preserve the coherence with Sections 1.1.2, 1.2.2, where algebraic structures have been defined in mathematical terms, implemented in Kenzo and in Isabelle. Second, because some alternatives to the implementation of functions will be given in Sections 3.7 and 3.8, and thus we considered that introducing the details about the implementation found in Isabelle also in the same chapter would ease readability.

Chapter 2

The Basic Perturbation Lemma

In this chapter a detailed proof of the Basic Perturbation Lemma, which can be seen as a tool for the computation of homology groups, will be given.

2.1 Statement of the BPL

As announced in Section 1.1.1, a particular case of homotopy equivalence is introduced here:

Definition 2.1.1. Given two chain complexes (D_*, d_{D_*}) and (C_*, d_{C_*}), a *reduction* between them is a triple of homomorphisms $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ satisfying:

1. The components f and g are chain complex homomorphisms of degree 0, $f : (D_*, d_{D_*}) \to (C_*, d_{C_*})$ and $g\colon (C_*, d_{C_*}) \to (D_*, d_{D_*})$.
2. The component h is a homotopy operator on D_*, that is to say, a graded group endomorphism of degree +1.
3. And the following relations hold:
   (a) $fg = \mathrm{id}_{C_*}$;
   (b) $gf + d_{D_*}h + hd_{D_*} = \mathrm{id}_{D_*}$;
   (c) $fh = 0_{\mathrm{Hom}(D_*,C_*)}$;
   (d) $hg = 0_{\mathrm{Hom}(C_*,D_*)}$;
   (e) $hh = 0_{\mathrm{End}(D_*)}$.

Definition 2.1.2. Let (D_*, d_{D_*}) be a chain complex. A *perturbation* of the differential d_{D_*} is a graded group endomorphism $\delta_{D_*} : D_* \to D_*$ (degree -1) such that $d_{D_*} + \delta_{D_*}$ is a differential for the graded group D_*. A perturbation δ_{D_*} of d_{D_*} satisfies the *nilpotency condition* with respect to a reduction $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$, whenever the composition $\delta_{D_*}h$ is pointwise nilpotent, that is, given x an element of D_*, there exists a natural number n such that $(\delta_{D_*}h)^n(x) = 0$, where n can depend on each x in D_*. The same property also holds for the composition $h\delta_{D_*}$.

Now, the definition of the *degree of nilpotency* is introduced.

Definition 2.1.3. Let f be an endomorphism of a chain complex (D_*, d_{D_*}) satisfying the nilpotency condition, that is, for every $x \in D_*$ there is an integer $n > 0$ verifying that $f^n(x) = 0_{D_*}$. Then, the minimum integer n (which depends on x) satisfying that $f^n(x) = 0_{D_*}$ is called the *degree of nilpotency for f and x*, and will be denoted by $n_f(x)$.

Note that if $n_f(x)$ is the degree of nilpotency of x for an endomorphism $f\colon (D_*, d_{D_*}) \to (D_*, d_{D_*})$, obviously $f^m(x) = 0_{D_*}$ for all $m \geq n_f(x)$.

The following proposition is needed to ensure the soundness of the statement of the Basic Perturbation Lemma.

Proposition 2.1.4. *Let $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ be a reduction between two chain complexes and δ_{D_*} a perturbation of d_{D_*} satisfying the nilpotency condition with respect to the reduction. Then both $\sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$ and $\sum_{i=0}^{\infty}(-1)^i(h\delta_{D_*})^i$ denote endomorphisms of degree 0 of the graded group D_*.*

Proof. First, since δ_{D_*} satisfies the nilpotency condition with respect to (f, g, h), the series $\sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$ denotes always a finite sum, when applied to a given element x in D_*.

Let ϕ be $\sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$. Let $x, y \in D_i$, and let $+_{D_i}$ be the additive operation of the i-th abelian group of the graded group D_*. From the definition of endomorphism, it must be verified that $\phi(x +_{D_i} y) = \phi(x) +_{D_i} \phi(y)$.

In order to do so, it can be observed that the degree of h is $+1$ and also from the definition of perturbation, the degree of δ_{D_*} is -1. Therefore, the degree of the combinations $h\delta_{D_*}$ and $\delta_{D_*}h$ is the sum of both degrees, which in this case yields 0; now, the degree of $(\delta_{D_*}h)^n$ and $(h\delta_{D_*})^n$ with $n \in \mathbb{N}$ will be also 0. Taking into account that δ_{D_*} satisfies the condition of local nilpotency with respect to (f, g, h), there must exist a natural number $n_{\delta_{D_*}h}(x +_{D_i} y)$ such that $(\delta_{D_*}h)^j(x +_{D_i} y) = 0_{D_i}$, $\forall j \geqslant n_{\delta_{D_*}h}(x +_{D_i} y)$. There must be also natural numbers $n_{\delta_{D_*}h}(x)$ and $n_{\delta_{D_*}h}(y)$ such that $\phi^{n_{\delta_{D_*}h}(x)}(x) = 0_{D_i}$ and $\phi^{n_{\delta_{D_*}h}(y)}(y) = 0_{D_i}$. Let us define $k = \max\{n_{\delta_{D_*}h}(x +_{D_i} y), n_{\delta_{D_*}h}(x), n_{\delta_{D_*}h}(y)\}$, and then:

$$\begin{aligned}
\phi(x +_{D_i} y) &= \sum_{j=0}^{k}(-1)^j(\delta_{D_*}h)^j(x +_{D_i} y)\\
&= \sum_{j=0}^{k}(-1)^j(\delta_{D_*}h)^j(x) +_{D_i} \sum_{j=0}^{k}(-1)^j(\delta_{D_*}h)^j(y)\\
&= \phi(x) +_{D_i} \phi(y)
\end{aligned}$$

From the definition of ϕ is also clear that, for every degree i, $\phi(0_{D_i}) = 0_{D_i}$. Thus, ϕ is an endomorphism.

The same ideas can be also applied to $\sum_{i=0}^{\infty}(-1)^i(h\delta_{D_*})^i$, once we have proved that the composition $h\delta_{D_*}$ satisfies the nilpotency condition. Let x be in D_*; from the nilpotency condition satisfied by $\delta_{D_*}h$, there must exist a i such that $(\delta_{D_*}h)^i(\delta_{D_*}(x)) = 0_{D_*}$. Now we choose a number j greater than i. The composition $(h\delta_{D_*})^j(x)$ can be seen as $h(\delta_{D_*}h)^{j-1}(\delta_{D_*}(x))$, which, taking into account that $j-1$ is greater than or equal to i, is equal to 0_{D_*}. The remaining part of the proof is similar to the one given for $\delta_{D_*}h$. ■

The statement of the BPL is the following:

Theorem 2.1.5. Basic Perturbation Lemma. *Let $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ be a chain complex reduction and $\delta_{D_*}\colon D_* \to D_*$ a* perturbation *of the differential d_{D_*} satisfying the nilpotency condition with respect to the reduction (f, g, h). Then a new reduction $(f', g', h')\colon (D'_*, d_{D'_*}) \Rightarrow (C'_*, d_{D'_*})$ can be obtained where the underlying graded groups D_* and D'_* (resp. C_* and C'_*) are the same, but the differentials are perturbed: $d_{D'_*} = d_{D_*} + \delta_{D_*}, d_{C'_*} = d_{C_*} + \delta_{C_*}$, and $\delta_{C_*} = f\delta_{D_*}\psi g$; $f' = f\phi$; $g' = \psi g$; $h' = h\phi$, where $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$, and $\psi = \sum_{i=0}^{\infty}(-1)^i(h\delta_{D_*})^i$.*

The BPL is a central result in algorithmic homological algebra (in particular, it has been intensively used in the symbolic computation systems EAT [Rubio et al., 1997] and Kenzo [Dousson et al., 1999]). It first appears in [Shih, 1962] and it was rewritten in modern terms in [Brown, 1965]. Since then, plenty of proofs have been described in the literature (see, for instance, [Gugenheim, 1972], [Barnes and Lambe, 1991], [Rubio and Sergeraert, 1997]). In the next section we briefly illustrate the importance of the BPL.

2.1.1 Topological motivation

Although the BPL statement is established in purely algebraic terms, the very essence of the result stems from geometry, and more concretely from simplicial topology. The algebraic notion of *perturbation* is the counterpart of the geometric notion of *torsion*. To be precise, two spaces F (for *fiber*) and B (for *base*) can be glued together through a *twisting function* τ, giving rise to a new space, called *total space* and denoted by $F \times_\tau B$. This is the notion of *simplicial fibration*, combinatorial version of the topological concept of *fibre bundle*. (In this section, every notion and result can be found, except if explicitly stated, in [May, 1967].)

Let us explore first the case of a non-twisted product $F \times B$, that is to say, the case of the cartesian product of two spaces. The bridge between Geometry and Algebra (in other words, the key brick of Algebraic Topology) is the notion of *chain complex associated to a topological space*. If X is a space, its associated chain complex is usually denoted by $C(X)$. A natural question to be asked is which is the relation between the chain complex $C(F \times B)$ and the complexes $C(F)$ and $C(B)$. In the algebraic category of chain complexes, there exists a notion of product: *the tensor product*, denoted by the symbol $\otimes$. Then a well-known result due to Eilenberg and Zilber gives explicit formulas describing a reduction from $C(F \times B)$ to $C(F) \otimes C(B)$.

Now, the twisting function τ defining the fibration $F \times_\tau B$ from the cartesian product $F \times B$, also defines an algebraic *perturbation* on the chain complex $C(F \times B)$. This perturbation is locally nilpotent with respect to the Eilenberg-Zilber reduction, and therefore the BPL can be applied. In this way, the *twisted Eilenberg-Zilber theorem* is proved, giving a reduction from $C(F \times_\tau B)$ to $C(F) \otimes_t C(B)$, that is to say, to a *perturbed* tensor product of $C(F)$ and $C(B)$. In this simple way, the BPL enables to describe the homological information of the total space of a fibration in terms of the chain complexes of the fiber and the base space.

This appealing example of the BPL power can be considered, in fact, as the result leading to the very discovering of the perturbation machinery. Historically, this technique seems to be found for the first time, in an unstructured way, by Shih in [Shih, 1962], just to deal with the twisted version of the Eilenberg-Zilber theorem. Later on, the BPL has been systematized by R. Brown [Brown, 1965], and independently by Gugenheim [Gugenheim, 1972], always to cope with this Eilenberg-Zilber application.

2.1.2 Algorithmic interpretation

The relevant part from the Computer Algebra perspective is the algorithmic interpretation of the BPL. To explain this issue, certain preliminaries are needed.

A central problem in computational Algebraic Topology is to calculate, by means of a computer, *homology groups* of topological spaces. By definition, the homology group of a space X is the one of its associated chain complex $C(X)$. In the case where X is presented as a simplicial set and there are finitely many non-degenerate simplexes at each dimension, $C(X)$ can be described as a graded free abelian group with finitely many generators at each degree. In this case, computing each homology group can be translated to a problem of diagonalizing certain integer matrices. So, we can assert that homology groups are computable in this finite type case.

Nevertheless, things become more interesting in Algebraic Topology when a space X is not of finite type (in the previously evoked sense) but it is known that its homology groups *are* of finite type. Then, it is natural to study if these homology groups are computable. Sergeraert introduced by 1985 a theory, called *effective homology*, in order to establish a framework where this computability question can be systematically studied.

For the sake of simplicity, we consider that a space X has *effective homology* if a reduction from $C(X)$ to a finite type chain complex is explicitly known. (This is a simplified version, since the more general notion of *strong homotopy equivalence* should be considered; in addition, very precise computability conditions must be imposed to each ingredient in the definition; see [Rubio and Sergeraert, 2002].)

Since if there exists a reduction between two chain complexes, the corresponding homology groups are isomorphic (see Theorem 1.1.9), several consequences follow from the definition of X as a space with effective homology. First, the homology groups of X are of finite type. Second, these homology groups are computable, *independently of the finite or infinite nature of the initial space* X.

Thus, the fact of determining that a space is with effective homology provides an algorithm to compute its homology groups. This is the strategy systematically used in the Kenzo system.

Now, we can look again at the BPL. It is worth noting that in the chain complex (C_*, d_{C_*}) (this chain complex will be usually named the "small" one of the reduction, being (D_*, d_{D_*}) the "big" one) only the differential is changed after the perturbation process. In particular, if C_* was of finite type in the input, then the corresponding chain complex will continue to be of finite type in the output. In other words, the BPL can be interpreted as an algorithm computing a chain complex with effective homology from another chain complex with effective homology (plus a locally nilpotent perturbation). The reason for this result is that the statement of the BPL itself describes the manner of constructing in an algorithmic way (by means of the series $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$) the ingredients of the output reduction. By using the terminology of Chapter 4, we can say that the BPL has a *constructive statement.*

2.1.3 Application to Computer Algebra in Algebraic Topology

As applications of this algorithmic interpretation of the BPL, it turns out that the most usual *spectral sequences* in Algebraic Topology can be converted (through convenient BPL instances) into usable algorithms. Indeed, the effective *Serre spectral sequence* allows the computer to calculate the effective homology of the total space $F \times_\tau B$ of a fibration, in terms of the effective homology of F and B [Rubio and Sergeraert, 2002]. Just as another example, the effective *Eilenberg-Moore spectral sequence* computes the effective homology of the fiber F from the effective homology of $F \times_\tau B$ and B [Rubio and Sergeraert, 1988]. These effective versions of spectral sequences, as implemented in Kenzo [Dousson et al., 1999], are used to compute the homology of iterated loop spaces and some homotopy groups.

To conclude, the BPL is the most important algorithm in Kenzo (as the most frequent in central parts). It allows the programmer to work in a very high conceptual level, quite close to the standard way of working of topologists. Finally, both the complexity of the code (see the corresponding Kenzo fragment in Chapter 4, Figure 4.1) and its relevance from a practical point of view justify the decision of choosing the BPL as a case study for our approach to mechanized reasoning in Algebraic Topology.

2.2 A detailed proof of the BPL

As it has been just explained, the Basic Perturbation Lemma is a central result in Homological Algebra and it can be also interpreted as an algorithm which offers a constructive method to determine homology groups of chain complexes (linking infinite chain complexes with chain complexes of finite type).

At a first sight, it could be thought that, as far as the theorem defines how to build a new reduction between two chain complexes, the proof could just try to check that this new reduction satisfies the conditions of reduction stated in Definition 2.1.1. Nevertheless, as the literature has shown, the direct approach is too complicated.

The proof exposed in this section follows the one in [Rubio and Sergeraert, 1997], and has two different parts; the first one is devoted to the work with the series (that could be also named the "analytic" part), and the second one mainly works with structures and properties from Homological Algebra (and could be defined as the "equational" part). By introducing the identities obtained in the first part of the proof in the second part, a complete proof of the BPL is obtained. The second part of the proof is divided into six lemmas which combined in a suitable way produce the complete proof of the BPL.

Part 1. *Let ψ be the series $\sum_{i=0}^{\infty}(-1)^i(h\delta_{D*})^i$. From the BPL hypotheses, the following equations are proved: $\phi h = h\psi$; $\delta_{D*}\phi = \psi\delta_{D*}$; $\phi = \mathrm{id}_{D*} - h\delta_{D*}\phi = \mathrm{id}_{D*} - \phi h\delta_{D*} = \mathrm{id}_{D*} - h\psi\delta_{D*}$; $\psi = \mathrm{id}_{D*} - \delta_{D*}h\psi = \mathrm{id}_{D*} - \psi\delta_{D*}h = \mathrm{id}_{D*} - \delta_{D*}\phi h$.*

Part 2. *Then, and* by only using the previous equations for ϕ and ψ*, the BPL conclusion is proved.*

The reason to divide the proof into two parts is the very different nature of the proofs that appear in the lemmas of each part. The proofs in Section 2.2.1, corresponding to Part 1, are based on handling of formal series. On the other hand, the proofs of the lemmas in Section 2.2.2, corresponding to Part 2, are of algebraic nature, and they involve Homological Algebra structures, such as chain complexes and reductions, that can be avoided in the first part. Following these ideas, we will introduce in Section 2.2.1 four lemmas expressing identities between homomorphisms and formal series which will be needed in Section 2.2.2. In Section 2.2.2, six lemmas related to Homological Algebra using the identities proved in Section 2.2.1 will be proved, which will be finally combined to produce a structured proof of the BPL.

This separation will allow us to deal with different problems in the different stages of the later implementation of our proof in a theorem prover. Some reasons have led us to propose a so detailed proof of every lemma, much more detailed than in the usual style these proofs would have in a standard mathematical text:

1. A proof as close as possible to the one that will be introduced (or implemented) in the theorem prover is intended. Details about how mathematical objects are implemented in a concrete theorem prover will not be commented on in this chapter,

and we will focus our attention on them later, but every non trivial step of the proof will have associated a procedure (in the form of a proposition or a previous result) useful to reach the needed goal (the following step).

2. This semi-formalized style of presenting proofs will be helpful to estimate the complexity of the formalized proofs in terms of simple steps.

3. The mathematical proofs will give us an approximation of how close we are to the theorem proving languages based on tactics, offering us valuable information to determine which language is more suitable for our work.

4. The mathematical proofs and the steps of these mathematical proofs will determine to some extent the infrastructure needed in the theorem prover, that we will have to produce in a later stage of our work. Libraries will have to be created (or maybe adapted from the existing libraries in the theorem prover) depending on what kind of reasoning we need in our proofs.

5. These highly detailed proofs will offer information about some steps that usually are forgotten in the proofs of the BPL in the literature and that are needed when a mechanized proof is intended.

6. The detailed mathematical proofs are useful to detect the possible problems appearing later in the implementation in the theorem prover, and also allow us to take in advance some decisions about the design of the structures involved in the proofs.

In the following sections of this chapter, we intend to produce a mathematical proof of the BPL satisfying all these premises; some comments will be added to remark when these goals have been reached.

2.2.1 A detailed proof of the BPL: the series

The lemmas and proofs appearing in this section are mostly based on formal infinite series. Two functional series whose terms satisfy a nilpotency condition are introduced; these two series define two homomorphisms between graded groups and the goal will be to prove some identities where these homomorphisms appear. The properties of the homomorphisms, and not the formal series, will be used for the second part of the proof of the BPL (see Section 2.2.2) in order to define the output of the algorithm associated to the BPL, i.e., a reduction between chain complexes[1].

The definitions introduced in Sections 1.1.1 and 2.1 will be used in the following statements and proofs. The complete proof of the BPL will be given later in Lemma 2.2.20 after the introduction of the preliminary lemmas. Now, Lemmas 2.2.1 and 2.2.2 will be proved; their results will be used for proving Lemmas 2.2.3 and 2.2.4.

[1]As it was mentioned before, the BPL can be seen as an algorithm with a reduction and a perturbation as input and a new reduction as output.

Lemma 2.2.1. *Given a reduction $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ and a perturbation δ_{D_*} of the differential d_{D_*} satisfying the nilpotency condition with respect to the reduction (f, g, h), the degree of nilpotency of every $x \in D_*$ for $\delta_{D_*}h$, $n_{\delta_{D_*}h}(x)$, is greater than or equal to the degree of nilpotency of $h(x)$ for $h\delta_{D_*}$, $n_{h\delta_{D_*}}(h(x))$.*

Proof. Let x be an element of D_*; let us consider $h(x)$, which is also an element of D_*, and let $n_{\delta_{D_*}h}(x)$ be the degree of nilpotency of x for $\delta_{D_*}h$ and $n_{h\delta_{D_*}}(h(x))$ the degree of nilpotency for $h\delta_{D_*}$ of $h(x)$. We prove that whenever $(\delta_{D_*}h)^{n_{\delta_{D_*}h}(x)}(x) = 0_{D_*}$, then $(h\delta_{D_*})^{n_{\delta_{D_*}h}(x)}(h(x)) = 0_{D_*}$:

$$\begin{aligned}
(h\delta_{D_*})^{n_{\delta_{D_*}h}(x)}(h(x)) &= h(\delta_{D_*}h)^{n_{\delta_{D_*}h}(x)}(x) && \text{(by applying associativity)}\\
&= h(0_{D_*}) && \text{(from the premises)}\\
&= 0_{D_*} && \text{(since h is a endomorphism)}
\end{aligned}$$

And therefore, the coefficient $n_{h\delta_{D_*}}(h(x))$ is smaller than or equal to $n_{\delta_{D_*}h}(x)$. ■

The hypotheses "let $(f, g, h) : (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ be a reduction between chain complexes" and "let δ_{D_*} be a perturbation satisfying the nilpotency condition with respect to the reduction (f, g, h)", introduced in Lemma 2.2.1 and in the BPL statement are also needed in most of the results of this section, and we will abbreviate them by simply writing "under the BPL hypotheses".

Lemma 2.2.2. *Under the BPL hypotheses, the degree of nilpotency of every $x \in D_*$ for $h\delta_{D_*}$, $n_{h\delta_{D_*}}(x)$, is greater than or equal to the degree of nilpotency $n_{\delta_{D_*}h}(\delta_{D_*}(x))$ for $\delta_{D_*}h$.*

Proof. Let x be an element of D_*; let us consider $\delta_{D_*}(x)$, which is also an element of D_*, and let $n_{\delta_{D_*}h}(\delta_{D_*}(x))$ be the degree of nilpotency of $\delta_{D_*}(x)$ for $\delta_{D_*}h$ and $n_{h\delta_{D_*}}(x)$ the degree of nilpotency for $h\delta_{D_*}$ of x. We prove that whenever $(h\delta_{D_*})^{n_{h\delta_{D_*}}(x)}(x) = 0_{D_*}$, then $(\delta_{D_*}h)^{n_{h\delta_{D_*}}(x)}(\delta_{D_*}(x)) = 0_{D_*}$:

$$\begin{aligned}
(\delta_{D_*}h)^{n_{h\delta_{D_*}}(x)}(\delta_{D_*}(x)) &= \delta_{D_*}(h\delta_{D_*})^{n_{h\delta_{D_*}}(x)}(x) && \text{(by applying associativity)}\\
&= \delta_{D_*}(0_{D_*}) && \text{(from the premises)}\\
&= 0_{D_*} && \text{(since h is a endomorphism)}
\end{aligned}$$

Therefore, the coefficient $n_{h\delta_{D_*}}(x)$ is greater than or equal to $n_{\delta_{D_*}h}(\delta_{D_*}(x))$. ■

The lemmas we are now introducing will allow us to produce the proofs for the lemmas in Section 2.2.2 avoiding the presence of formal series. They express some equalities between graded group homomorphisms; another reason to extract them from the second part of the proof of the BPL that we will give in Section 2.2.2 is that explicit reference to concrete structures of Homological Algebra appears neither in their statements, nor in their proofs.

Let ψ and ϕ be $\sum_{i=0}^{\infty}(-1)^i(h\delta_{D*})^i$ and $\sum_{i=0}^{\infty}(-1)^i(\delta_{D*}h)^i$, respectively, as defined in the BPL statement.

Lemma 2.2.3. *Under the BPL hypotheses, $\psi h = h\phi$.*

Proof. Let x be an element of $D*$. We have already proved in Proposition 2.1.4 that the formal series defined by both ψ and ϕ are endomorphisms of the graded group $D*$, and h is an endomorphism of $D*$ too. We prove that, for any x in the graded group $D*$, the two compositions produce the same result:

$$\psi h(x) = (\sum_{i=0}^{\infty}(-1)^i(h\delta_{D*})^i)h(x) \tag{2.1}$$

$$\begin{aligned}
&= h(x) - (h\delta_{D*})h(x) + \ldots + (-1)^{n_{h\delta_{D*}}(h(x))}(h\delta_{D*})^{n_{h\delta_{D*}}(h(x))}(h(x)) &(2.2)\\
&= h(x) - h(\delta_{D*}h)(x) + \ldots + (-1)^{n_{h\delta_{D*}}(h(x))}\ \ h(\delta_{D*}h)^{n_{h\delta_{D*}}(h(x))}(x) &(2.3)\\
&= h\phi(x) &(2.4)
\end{aligned}$$

Identity 2.1 is obtained by unfolding the definition of ψ.

Identity 2.2 is obtained thanks to the condition of local nilpotency applied to $h(x)$.

Identity 2.3 is obtained by associativity.

Identity 2.4 is obtained from the definition of ϕ and introducing Lemma 2.2.1. ■

Lemma 2.2.4. *Under the BPL hypotheses, $\delta_{D*}\psi = \phi\delta_{D*}$.*

Proof. We apply the same idea introduced in the previous proof:

$$\begin{aligned}
\delta_{D*}\psi(x) &= \delta_{D*}(\sum_{i=0}^{\infty}(-1)^i(h\delta_{D*})^i)(x) &(2.5)\\
&= \delta_{D*}(x) - \delta_{D*}(h\delta_{D*})(x) + \ldots + \delta_{D*}(-1)^{n_{h\delta_{D*}}(x)}(h\delta_{D*})^{n_{h\delta_{D*}}(x)}(x) &(2.6)\\
&= \delta_{D*}(x) - (\delta_{D*}h)\delta_{D*}(x) + \ldots + (-1)^{n_{h\delta_{D*}}(x)}(\delta_{D*}h)^{n_{h\delta_{D*}}(x)}\delta_{D*}(x) &(2.7)\\
&= \phi\delta_{D*}(x) &(2.8)
\end{aligned}$$

Identity 2.5 is obtained by unfolding the definition of ψ.

Identity 2.6 is obtained thanks to the condition of local nilpotency applied to x.

Identity 2.7 is obtained by associativity.

Identity 2.8 is obtained from the definition of ϕ and introducing Lemma 2.2.2. ■

The following lemmas will be also useful for the final proof of the BPL. They establish equational relations among the different endomorphisms (including the previously defined series ψ and ϕ) appearing in the statement of the BPL. Previous Lemmas 2.2.3 and 2.2.4 will have to be used in the proofs of the following ones:

Lemma 2.2.5. *Under the BPL hypotheses,*
$\psi = \mathrm{id}_{D*} - h\delta_{D*}\psi = \mathrm{id}_{D*} - \psi h\delta_{D*} = \mathrm{id}_{D*} - h\phi\delta_{D*}$.

Proof. Let x be an element of $D*$, and $n_{h\delta_{D*}}(x)$ and $n_{\delta_{D*}h}(x)$, respectively, the degree of nilpotency for $h\delta_{D*}$ and $\delta_{D*}h$ of x. Then we choose an integer $k > \max\{n_{h\delta_{D*}}(x), n_{\delta_{D*}h}(x)\}$:

$$\begin{aligned}
\psi(x) &= \sum_{i=0}^{k}(-1)^i(h\delta_{D*})^i(x) && \text{(unfolding the definition of } \psi\text{)}\\
&= \mathrm{id}_{D*}(x) - \sum_{i=1}^{k}(-1)^i(h\delta_{D*})^i(x) && \text{(extracting the first term)}\\
&= \mathrm{id}_{D*}(x) - (h\delta_{D*})\sum_{i=0}^{k-1}(-1)^i(h\delta_{D*})^i(x) && \text{(factorizing the general term)}\\
&= \mathrm{id}_{D*}(x) - h\delta_{D*}\psi(x) && \text{(introducing the definition of } \psi\\
& && \text{and } k > \max\{n_{h\delta_{D*}}(x), n_{\delta_{D*}h}(x)\})\\
&= \mathrm{id}_{D*}(x) - h\phi\delta_{D*}(x) && \text{(by Lemma 2.2.4)}\\
&= \mathrm{id}_{D*}(x) - \psi h\delta_{D*}(x) && \text{(by Lemma 2.2.3)}
\end{aligned}$$

The integer k has to be chosen strictly greater than both $n_{h\delta_{D*}}(x)$ and $n_{\delta_{D*}h}(x)$ because otherwise the series representing ψ or ϕ could be incomplete. We have proved that the four equations represent the same endomorphism of $D*$. ■

Lemma 2.2.6. *Under the BPL hypotheses,*
$\phi = \mathrm{id}_{D*} - \delta_{D*}h\phi = \mathrm{id}_{D*} - \phi\delta_{D*}h = \mathrm{id}_{D*} - \delta_{D*}\psi h$.

Proof. The proof is similar to the previous one; let x be an element of $D*$ and we choose an integer $k > \max\{n_{\delta_{D_*}h}(x), n_{h\delta_{D_*}}(x)\}$:

$$\begin{aligned}
\phi(x) &= \sum_{i=0}^{k}(-1)^i(\delta_{D_*}h)^i(x) && \text{(unfolding the definition of } \phi\text{)}\\
&= \mathrm{id}_{D_*}(x) - \sum_{i=1}^{k}(-1)^i(\delta_{D_*}h)^i(x) && \text{(extracting the first term)}\\
&= \mathrm{id}_{D_*}(x) - (\delta_{D_*}h)\sum_{i=0}^{k-1}(-1)^i(\delta_{D_*}h)^i(x) && \text{(factorizing the general term)}\\
&= \mathrm{id}_{D_*}(x) - \delta_{D_*}h\phi(x) && \text{(introducing the definition of } \phi\\
& && \text{and } k > \max\{n_{\delta_{D_*}h}(x), n_{h\delta_{D_*}}(x)\}\text{)}\\
&= \mathrm{id}_{D_*}(x) - \delta_{D_*}\psi h(x) && \text{(by Lemma 2.2.3)}\\
&= \mathrm{id}_{D_*}(x) - \phi\delta_{D_*}h(x) && \text{(by Lemma 2.2.4)}
\end{aligned}$$

Therefore, the four equations represent the same endomorphism of the graded group D_*. ■

The equalities obtained in Lemmas 2.2.3, 2.2.4, 2.2.5 and 2.2.6 are the only tool related to the series that will be used for the second part of the proof of the BPL. We have avoided the use of formal series for the rest of the proof, as will be seen when we develop the complete proof of the BPL in Lemma 2.2.20. Let us observe, in relation to the nature of the given proofs of Lemmas 2.2.3, 2.2.4, 2.2.5 and 2.2.6, that no explicit reference to the properties of the special algebraic structures appearing in the statements, such as chain complexes or graded groups, is needed. The proofs can be made in a generic setting, for instance, any ring of endomorphisms, that could be later instantiated to our concrete ring $\mathrm{End}(D_*)$.

2.2.2 A detailed proof of the BPL: the lemmas

In this second part of the proof of the BPL, the homomorphisms ϕ and ψ, representing the formal series from Section 2.2.1, will be used, as well as the properties we have proved in Lemmas 2.2.3, 2.2.4, 2.2.5 and 2.2.6.

The goal in this second part consists in, just with the equalities of the first part, building the new reduction $(f', g', h')\colon (D'_*, d_{D'_*}) \Rightarrow (C'_*, d_{C'_*})$ stated in the BPL. This will complete the proof of the BPL. The proof will be divided now into six different lemmas. These six lemmas have constructive statements (on this notion, see Chapter 4), in the sense that they produce intermediate results, which combined in a suitable way give a complete proof of the BPL. These "constructive" lemmas recall the algorithmic nature of the BPL. Three main reasons to divide the proof into the six lemmas can be mentioned:

1. To obtain a better understanding of the proof.

2. To produce lemmas with enough information to be meaningful by themselves, tackling the different difficulties of the proof separately.

3. To obtain guidelines to translate this part of the proof to a mechanized reasoning system.

The lemmas have been stated in a generic way (in the sense that they use as many premises as possible from the BPL) and the proofs have been produced in natural language but with a level of detail that tries to be close to the expected one in a theorem prover. The definitions and propositions needed for each lemma will be stated right before the lemma.

Definition 2.2.7. Let C_* be a graded group and $f \in \operatorname{End}(C_*)$ of degree 0; f is said to be a *projector* if $ff = f$.

Proposition 2.2.8. *Let C_* be a graded group and let $f \in \operatorname{End}(C_*)$ be a projector. Given $x \in \operatorname{im} f$, $f(x) = x$.*

Proof. Let x be an element of $\operatorname{im} f$; there exists a y in C_* such that $f(y) = x$; then $f(x) = ff(y) = f(y)$, since f is a projector; therefore, $f(x) = f(y)$ and finally, $f(x) = x$. ■

Proposition 2.2.9. *Given two chain complexes (D_*, d_{D_*}) and (C_*, d_{C_*}), and a chain complex homomorphism $f\colon (D_*, d_{D_*}) \to (C_*, d_{C_*})$, the set $\ker f$ together with the inherited operations from (D_*, d_{D_*}) is also a chain complex.*

Proof. First, it must be proved that $\ker f$ is closed under the operations on D_*. Let k be the degree of the homomorphism f. Given any x, $y \in \ker f_i$, let $+_{D_i}$ be the binary operation of the i-th abelian group of the chain complex (D_*, d_{D_*}). Then, by using that f_i is an abelian group homomorphism, $f_i(x +_{D_i} y) = f_i(x) +_{C_{i+k}} f_i(y) = 0_{C_{i+k}} +_{C_{i+k}} 0_{C_{i+k}} = 0_{C_{i+k}}$. It is also clear from the definition of homomorphism that $f_i(0_{D_i}) = 0_{C_{i+k}}$.

Now, in order to have a chain complex structure, it must be also seen that d_{D_*} is a differential for the set $\ker f$ with the operations of D_*, so it must be an endomorphism of $\ker f$ satisfying that $\forall x \in \ker f$, $d_{D_*}(d_{D_*}(x)) = 0_{D_*}$.

First, d_{D_*} is an endomorphism of $\ker f$ since given any $x \in \ker f_i$, $f_i d_{D_{i+k}}(x) = d_{C_{i+k}} f_i(x)$ and being $x \in \ker f_i$, this is $d_{C_{i+k}}(0_{C_{i+k}}) = 0_{C_{i+k}}$ from the properties of d_{C_*} like endomorphism of the graded group C_*.

The differential condition is satisfied as far as $\ker f \subseteq D_*$, and then every element in $\ker f$ will be such that $d_{D_*}(d_{D_*}(x)) = 0_{D_*}$. Then $(\ker f, d_{D_*})$ is a chain complex. ■

Remark. *In the previous proof, d_{D_*} denotes both the differential of the graded group D_* and also the differential for the graded subgroup $\ker f$ of D_*. In the second case, the differential really refers to $(d_{D_*})|_{(\ker p)}$, but this notation is too verbose and could be misleading, being too different from usual notation in mathematical texts. Therefore,* *we will use the notation displayed in the previous proof. Nevertheless, behind this notation there is a major question when implementing proofs in a theorem prover. When a homomorphism is represented in a theorem prover with its domain and codomain, modifying them requires special tools and explicit processes, which are skipped in the standard mathematical presentations. This will be explicitly treated in our third approach (see Section 3.7).*

Proposition 2.2.10. *Given two chain complexes* (D_*, d_{D_*}) *and* (C_*, d_{C_*}) *and a chain complex homomorphism* $f\colon (D_*, d_{D_*}) \to (C_*, d_{C_*})$*, the set* $\operatorname{im} f$ *together with the inherited operations of* (C_*, d_{C_*}) *is a chain complex.*

Proof. First, it must be proved that $\operatorname{im} f$ is a closed set under the operations of C_*. Let k be the degree of the homomorphism f. Given any x, y in $\operatorname{im} f_{i+k}$, let $+_{C_{i+k}}$ be the inner operation of the underlying abelian group of the chain complex (C_*, d_{C_*}). From the definition of the set $\operatorname{im} f$, there should exist x', y' in D_i such that $f_i(x') = x$ and $f_i(y') = y$. Now in order to see that $+_{C_*}$ is closed we have that $x +_{C_{i+k}} y = f_i(x') +_{C_{i+k}} f_i(y') = f_i(x' +_{D_i} y')$. Therefore, $x +_{C_{i+k}} y \in \operatorname{im} f_{i+k}$.

In order to have a chain complex structure, it must be seen that d_{C_*} is also a differential for the set $\operatorname{im} f$, so it must be an endomorphism of the graded group $\operatorname{im} f$ such that $d_{D_*} d_{D_*} = 0_{\operatorname{End}(D_*)}$.

First, it is an endomorphism of $\operatorname{im} f$, since given any element x in $\operatorname{im} f_{i+k}$, let us consider an element x' in D_i such that $f_i(x') = x$; then it follows that $d_{C_{i+k}}(x) = d_{C_{i+k}} f_i(x') = f_i d_{D_i}(x')$, and being $d_{D_i}(x')$ in D_i, $f_i d_{D_i}(x')$ again is an element of $\operatorname{im} f_{i+k}$, and therefore d_{C_*} is an endomorphism for $\operatorname{im} f$.

The differential condition is proved provided that $\operatorname{im} f \subseteq C_*$, and then for every element in $\operatorname{im} f$ will be satisfied that $d_{D_*} d_{D_*}(x) = 0_{D_*}$. Then $(\operatorname{im} f, d_{C_*})$ is a chain complex (in the following, notation proposed in the previous remark will be used), and a chain subcomplex of (C_*, d_{C_*}) . ■

Lemma 2.2.11. *Let* $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ *be a chain complex reduction. Then, there exists a canonical and explicit chain complex isomorphism between* (D_*, d_{D_*}) *and the direct sum* $(\ker gf, d_{D_*}) \oplus (C_*, d_{C_*})$*. In particular,* $\omega\colon (C_*, d_{C_*}) \to (\operatorname{im} gf, d_{D_*})$ *defined by* $x \mapsto g(x)$ *and* $\omega^{-1}\colon (\operatorname{im} gf, d_{D_*}) \to (C_*, d_{C_*})$*, defined by* $x \mapsto f(x)$*, are inverse isomorphisms of chain complexes.*

Proof. In the proof π will denote the composition gf; first, it can be observed that π is a projector. From Definition 2.1.1, if (f, g, h) is a reduction from (D_*, d_{D_*}) to (C_*, d_{C_*}), then $fg = \operatorname{id}_{C_*}$, and therefore $\pi\pi = gfgf = g(fg)f = \pi$; this property will be used later in the proof.

From Propositions 2.2.9 and 2.2.10, it can be observed that both $(\ker \pi, d_{D_*})$ and $(\operatorname{im} \pi, d_{D_*})$ are chain complexes, since $\pi\colon (D_*, d_{D_*}) \to (D_*, d_{D_*})$ is a homomorphism between chain complexes (π is the composition of two such homomorphisms).

Now, there is an isomorphism

$$\sigma\colon (D_*, d_{D_*}) \to (\ker\pi, d_{D_*}) \oplus (\operatorname{im}\pi, d_{D_*})$$

defined by $\sigma\colon x \mapsto ((d_{D_*}h + hd_{D_*})(x), \pi(x))$ which has as inverse isomorphism σ^{-1}, defined as the sum of the two components of a pair, $(x,y) \mapsto x+y$. We will now prove that both compositions $\sigma\sigma^{-1}$ and $\sigma^{-1}\sigma$ are equal to the identity, and therefore they define a bijection between (D_*, d_{D_*}) and $(\ker\pi, d_{D_*}) \oplus (\operatorname{im}\pi, d_{D_*})$ considered as sets. Later it will be proved that σ is a chain complex isomorphism.

1. $\sigma^{-1}\sigma = \mathrm{id}_{(D_*, d_{D_*})}$; given $x \in D_*$

$$\begin{aligned}
\sigma^{-1}\sigma(x) &= \sigma^{-1}((d_{D_*}h + hd_{D_*})(x), gf(x)) && \text{(from the definition of } \sigma\text{)}\\
&= (d_{D_*}h + hd_{D_*})(x) + gf(x) && \text{(from the definition of } \sigma^{-1}\text{)}\\
&= \mathrm{id}_{(D_*, d_{D_*})}(x) && \text{(by applying the reduction prop. (b)}\\
& && \text{stated in Definition 2.1.1)}
\end{aligned}$$

2. $\sigma\sigma^{-1} = \mathrm{id}_{(\ker\pi, d_{D_*})\oplus(\operatorname{im}\pi, d_{D_*})}$; given $x \in (\ker\pi, d_{D_*}) \oplus (\operatorname{im}\pi, d_{D_*})$, $x = (x_1, x_2)$,

$$\begin{aligned}
(\sigma\sigma^{-1})(x_1,x_2) &= \sigma(x_1+x_2) && \text{(from the definition of } \sigma^{-1}\text{)}\\
&= ((d_{D_*}h + hd_{D_*})(x_1+x_2), \pi(x_1+x_2)) && \text{(from the definition of } \sigma\text{)}\\
&= ((d_{D_*}h + hd_{D_*})(x_1+x_2), \pi(x_2)) && \text{(because } x_1 \in \ker\pi\text{)}\\
&= ((d_{D_*}h + hd_{D_*})(x_1+x_2), x_2) && \text{(by applying Proposition 2.2.8)}\\
&= (x_1 + (d_{D_*}h + hd_{D_*})(x_2), x_2) && \text{(from } \pi(x_1) = 0_{D_*}\\
& && \text{and Definition 2.1.1, prop. (b))}\\
&= (x_1, x_2) && \text{(from Proposition 2.2.8 with}\\
& && \pi(x_2) + (d_{D_*}h + hd_{D_*})(x_2) = x_2\text{)}
\end{aligned}$$

Thus, the equalities $\sigma\sigma^{-1} = \mathrm{id}_{(\ker\pi, d_{D_*})\oplus(\operatorname{im}\pi, d_{D_*})}$ and $\psi^{-1}\psi = \mathrm{id}_{(D_*, d_{D_*})}$ hold and the bijection has been defined.

Now it should be proved that σ is a chain complex isomorphism. From its definition, it can be seen that it is at least a graded group homomorphism, since it is defined like a combination of the graded group homomorphisms h and d_{D_*}. In order to prove that σ is a chain complex homomorphism from (D_*, d_{D_*}) to $(\ker\pi, d_{D_*}) \oplus (\operatorname{im}\pi, d_{D_*})$, it has to be proved that it is coherent with the differentials:

$$\begin{aligned}
\sigma d_{D*} &= (d_{D*}h + hd_{D*}, \pi)d_{D*} && \text{(from the definition of } \psi\text{)}\\
&= ((d_{D*}h + hd_{D*})d_{D*}, \pi d_{D*})\\
&= (d_{D*}hd_{D*}, gfd_{D*}) && \text{(from the properties of } d_{D*}\text{)}\\
&= (d_{D*}d_{D*}h + d_{D*}hd_{D*}, gd_{C*}f) && \text{(because the differential is coherent with}\\
& && \text{chain complex homomorphisms)}\\
&= (d_{D*}(d_{D*}h + hd_{D*}), d_{D*}gf) && \text{(same as before)}\\
&= (d_{D*}, d_{D*})((d_{D*}h + hd_{D*}), gf)\\
&= (d_{D*}, d_{D*})\sigma
\end{aligned}$$

The inverse σ^{-1} also is coherent with the differentials making use of the same properties, and so is also a chain complex homomorphism.

Now, by proving that there exists an isomorphism between the chain complexes $(\operatorname{im}\pi, d_{D*})$ and (C_*, d_{C*}), we will be able to finally define the isomorphism between the chain complexes (D_*, d_{D*}) and $(\ker\pi, d_{D*}) \oplus (C_*, d_{C*})$, just by composition of the two isomorphisms. Let us denote this isomorphism as ω^{-1}. Then it can be proved that

$$\omega\colon (C_*, d_{C*}) \to (\operatorname{im}\pi, d_{D*})$$

defined by $\omega = g$ is actually an isomorphism between chain complexes. The homomorphism ω will be also seen during the proof as $\omega = gfg$ because from the reduction properties of $(f, g, h)\colon (D_*, d_{D*}) \Rightarrow (C_*, d_{C*})$ given in Definition 2.1.1, $fg = \mathrm{id}_{C*}$. For instance, due to this property, the previous homomorphism is well defined, since $\omega = \pi g$ and then $\operatorname{im}\omega \subseteq \operatorname{im}\pi$. The inverse of ω will be defined as the restriction of f to $\operatorname{im}\pi$, $\omega^{-1} = f\big|_{\operatorname{im}\pi}$.

In the definition of ω^{-1}, let us observe the presence of the set $\operatorname{im}\pi$, whose definition, in general terms, is based on existential quantification, which in several cases is far from constructive mathematics. The selection of an element of the image set and the use of its preimage will be a common argument along the proofs, and implies that later will have to be translated into a theorem proving tool in order to formalize the proof. Thus, working with structures defined in terms of existential quantifiers can influence the choice of a theorem prover.

Now the identities $\omega\omega^{-1} = \mathrm{id}_{(\operatorname{im}\pi, d_{D*})}$ and $\omega^{-1}\omega = \mathrm{id}_{(C_*, d_{C*})}$ can be proved, using that π is a projector.

1. First, we prove that $\omega\omega^{-1} = \mathrm{id}_{(\operatorname{im}\pi, d_{D*})}$; given $x \in \operatorname{im}\pi$

$$\begin{aligned}
\omega\omega^{-1}(x) &= gf\big|_{\operatorname{im}\pi}(x)\\
&= gf\big|_{\operatorname{im}\pi}(\pi(y)) && \text{(for a } y \text{ in } D_*\text{)}\\
&= \pi\big|_{\operatorname{im}\pi}(\pi y) && \text{(from the definition of } \pi\text{)}\\
&= (\pi y) && \text{(since } y \text{ in } \operatorname{im}\pi \text{ and } \pi \text{ is a projector)}\\
&= x
\end{aligned}$$

2. Now, it can be also proved that $\omega^{-1}\omega = \mathrm{id}_{(C_*,d_{C_*})}$. Let us consider $x \in C_*$. (First we prove that $g(x)$ is an element of $\operatorname{im}\pi$, and then from the reduction property (a) the property holds).

$$\begin{aligned}
\omega^{-1}\omega(x) &= f\big|_{\operatorname{im}\pi} g(x) \\
&= f\big|_{\operatorname{im}\pi} g(fg(x)) && \text{(from the reduction property (a))} \\
&= f\big|_{\operatorname{im}\pi} (\pi)(g(x)) && \text{(from the definition of } \pi\text{)} \\
&= f\big|_{\operatorname{im}\pi} g(x) && \text{(Proposition 2.2.8 applied to } \pi \text{ and } g(x) \text{ in } \operatorname{im}\pi\text{)} \\
&= x && \text{(from the reduction property (a))}
\end{aligned}$$

Therefore, ω^{-1} and ω define an isomorphism between the graded groups $\operatorname{im}\pi$ and C_*. In order to prove that ω^{-1} is a chain complex isomorphism, it must also be coherent with respect to the differentials d_{C_*} and d_{D_*}; from the reduction premises, f is a chain complex homomorphism from (D_*, d_{D_*}) to (C_*, d_{C_*}), and therefore it satisfies that $d_{C_*}f = fd_{D_*}$. The only condition to be satisfied in order to prove that $d_{C_*}f\big|_{\operatorname{im}\pi} = f\big|_{\operatorname{im}\pi} d_{D_*}$ is that d_{D_*} is closed over the elements of $\operatorname{im}\pi$; this can be proved as follows. Let x be an element of $\operatorname{im}\pi$, and y such that $x = \pi(y)$:

$$\begin{aligned}
d_{D_*}(x) &= d_{D_*}(\pi(y)) \\
&= d_{D_*}(gf(y)) && \text{(unfolding the definition of } \pi\text{)} \\
&= gd_{C_*}f(y) && \text{(from the differential properties)} \\
&= gfd_{D_*}(y) && \text{(from the differential properties)} \\
&= \pi d_{D_*}(y) && \text{(from the definition of } \pi\text{)}
\end{aligned}$$

Finally, by composing the obtained chain complex isomorphisms $\sigma\colon (D_*, d_{D_*}) \to (\ker\pi, d_{D_*}) \oplus (\operatorname{im}\pi, d_{D_*})$ and $\omega^{-1}\colon (\operatorname{im}\pi, d_{D_*}) \to (C_*, d_{C_*})$, the isomorphism between chain complexes

$$(\mathrm{id}_{(\ker\pi, d_{D_*})}, \omega^{-1})\sigma : (D_*, d_{D_*}) \to (\ker gf, d_{D_*}) \oplus (C_*, d_{C_*})$$

is built. ■

Some technical results will be needed for the proof of the second lemma.

Proposition 2.2.12. *Let (D_*, d_{D_*}) be a chain complex, $h : D_* \to D_*$ (degree +1) a homomorphism of graded groups, satisfying $hh = 0_{\mathrm{End}(D_*)}$ and $hd_{D_*}h = h$. Let p be $d_{D_*}h + hd_{D_*}$ (degree 0). Then $(\ker p, d_{D_*})$ is a chain complex and a chain subcomplex of (D_*, d_{D_*}).*

Proof. From the premises it follows that h is a graded group endomorphism of D_*, and so is $p = d_{D_*}h + hd_{D_*}$. In order to satisfy the properties of a chain complex endomorphism,

p must be coherent with the differential d_{D_*}:

$$\begin{aligned} d_{D_*}p &= d_{D_*}(d_{D_*}h + hd_{D_*}) \\ &= d_{D_*}d_{D_*}h + d_{D_*}hd_{D_*} && \text{(by distributivity)} \\ &= d_{D_*}hd_{D_*} && \text{(since } d_{D_*} \text{ is a differential)} \\ &= d_{D_*}hd_{D_*} + hd_{D_*}d_{D_*} && \text{(same as before)} \\ &= (d_{D_*}h + hd_{D_*})d_{D_*} \\ &= pd_{D_*} \end{aligned}$$

Now that it has been proved that p is a chain complex homomorphism, applying Proposition 2.2.9 we obtain that $(\ker p, d_{D_*})$ is a chain complex.

Taking into account that $\ker p \subseteq D_*$, it follows that $(\ker p, d_{D_*})$ is a chain subcomplex of (D_*, d_{D_*}). ■

Proposition 2.2.13. *Let (D_*, d_{D_*}) be a chain complex and let p be an endomorphism of this chain complex. Let p (degree 0) be also a projector, i.e., $pp = p$. Then $\operatorname{im}(\mathrm{id}_{D_*} - p) \subseteq \ker p$.*

Proof. Let y be an element of the set $\operatorname{im}(\mathrm{id}_{D_*} - p)$. Then, there exists an element x in D_* (and here, we use again a non-constructive argument) such that $y = (\mathrm{id}_{D_*} - p)x$, and now:

$$\begin{aligned} p(\mathrm{id}_{D_*} - p)(x) &= p(x - p(x)) \\ &= p(x) - pp(x) && \text{(by distributivity)} \\ &= p(x) - p(x) && \text{(from definition of projector)} \\ &= 0_{D_*} \end{aligned}$$

Therefore, $(\mathrm{id}_{D_*} - p)(x) \in \ker p$, and the proof is complete. ■

Let p (degree 0) be the endomorphism of the chain complex (D_*, d_{D_*}) defined by $p = d_{D_*}h + hd_{D_*}$ and let us denote by $\mathrm{inc}_{\ker p}$ the canonical inclusion homomorphism $\mathrm{inc}_{\ker p} \colon \ker p \to D_*$ given by $x \mapsto x$. It is well defined since $\ker p$ is a chain subcomplex of D_* as we have proved in Proposition 2.2.12. Here it can be noted again the relevance of the domain and the codomain of homomorphisms, which forces us to denote in a distinguished way homomorphisms such as $\mathrm{id}_{(D_*, d_{D_*})} \colon (D_*, d_{D_*}) \to (D_*, d_{D_*})$, $\mathrm{id}_{(\ker p, d_{D_*})} \colon (\ker p, d_{D_*}) \to (\ker p, d_{D_*})$ and $\mathrm{inc}_{\ker p} \colon (\ker p, d_{D_*}) \to (D_*, d_{D_*})$.

Lemma 2.2.14. *Let (D_*, d_{D_*}) be a chain complex, h an endomorphism of the graded group D_* with degree $+1$, satisfying $hh = 0_{\mathrm{End}(D_*)}$ and $hd_{D_*}h = h$. Let p be $d_{D_*}h + hd_{D_*}$. Then p is a projector and, moreover, the triple $(\mathrm{id}_{D_*} - p, \mathrm{inc}_{\ker p}, h)$ defines a reduction from (D_*, d_{D_*}) to $(\ker p, d_{D_*})$.*

Proof. First let us see that, for every reduction (f_1, g_1, h_1) from a chain complex (D_*, d_{D_*}) to a chain complex (B_*, d_{B_*}), the homotopy operator h_1 satisfies the premises required in the statement about h. From the definition of homotopy operator, it follows that h_1 is an endomorphism with degree 0 such that $h_1h_1 = 0_{\mathrm{End}(D*)}$; now, in order to prove that $h_1 d_{D*} h_1 = h_1$, we recover the property $g_1 f_1 + h_1 d_{D*} + d_{D*} h_1 = \mathrm{id}_{(D*, d_{D*})}$. Then, by applying h_1 in both sides of the equality, we obtain $g_1 f_1 h_1 + h_1 d_{D*} h_1 + d_{D*} h_1 h_1 = h_1$; simplifying $h_1 h_1 = 0_{\mathrm{End}(D*)}$ and $f_1 h_1 = 0_{\mathrm{End}(B*)}$ (extracted from Definition 2.1.1 of reduction), the property is obtained. Therefore, this lemma can be also seen as an algorithm defining a reduction from a given one.

Now we prove that p is a projector. From Definition 2.2.7, we have to prove that p satisfies two premises:

1. It must be an endomorphism (degree 0) of $D*$, which follows from the definition of $p = d_{D*}h + hd_{D*}$, since both h (degree $+1$) and d_{D*} (degree -1) are endomorphisms of $D*$.

2. In addition to this, p must be idempotent; thus $pp = p$ has to be proved:

$$\begin{aligned} pp &= (d_{D*}h + hd_{D*})(d_{D*}h + hd_{D*}) && \text{(from the definition of } p) \\ &= d_{D*}hd_{D*}h + d_{D*}hhd_{D*} + hd_{D*}d_{D*}h + hd_{D*}hd_{D*} && \text{(unfolding the expressions)} \\ &= d_{D*}h + hd_{D*} && (\text{because } hh = 0_{\mathrm{End}(D*)}, \\ & && d_{D*}d_{D*} = 0_{\mathrm{End}(D*)}, \\ & && \text{and also } hd_{D*}h = h) \\ &= p \end{aligned}$$

In order to prove that $(\mathrm{id}_{D*} - p, \mathrm{inc}_{\ker p}, h)$ is a reduction from $(D*, d_{D*})$ to $(\ker p, d_{D*})$, and introducing Definition 2.1.1, the following must be proved. First, $(D*, d_{D*})$ is a chain complex from the premises, and so is $(\ker p, d_{D*})$ from Proposition 2.2.12. In addition to this, the following must be satisfied:

1. $(\mathrm{id}_{D*} - p)$ must be a homomorphism between the chain complexes $(D*, d_{D*})$ and $(\ker p, d_{D*})$ (degree 0), and $\mathrm{inc}_{\ker p}$ a homomorphism between the chain complexes $(\ker p, d_{D*})$ and $(D*, d_{D*})$ (degree 0):
   (a) From the definition of p and id_{D*} is clear that $(\mathrm{id}_{D*} - p)$ is a chain complex homomorphism of degree 0. In order to prove that it is well defined (i.e., its image contained in $\ker p$), Proposition 2.2.13 can be applied, once we know hat p is a projector.
   (b) From the definition of $\mathrm{inc}_{\ker p}$ it is derived that it is a chain complex homomorphism.

2. h must be an endomorphism of the graded group $D*$ (degree $+1$), which is among the premises.

3. The following relations between the chain complexes in the reduction must be satisfied:

(a) $(\mathrm{id}_{D*} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\ker p}$; given $x \in \ker p$,

$$\begin{aligned}(\mathrm{id}_{D*} - p)\,\mathrm{inc}_{\ker p}(x) &= x - p(x)\\ &= x \qquad \text{(applying that } x \in \ker p)\end{aligned}$$

and therefore, the composition is equal to the identity function restricted to $\ker p$.

(b) $\mathrm{inc}_{\ker p}(\mathrm{id}_{D*} - p) + d_{D*}h + hd_{D*} = \mathrm{id}_{D*}$; given $x \in D*$ and using that $p = d_{D*}h + hd_{D*}$,

$$\begin{aligned}(\mathrm{inc}_{\ker p}(\mathrm{id}_{D*} - p) + d_{D*}h + hd_{D*})(x) &= (\mathrm{inc}_{\ker p}(\mathrm{id}_{D*} - p) + p)(x)\\ &= \mathrm{inc}_{\ker p}(x - p(x)) + p(x)\\ &= x - p(x) + p(x) \quad \text{(because } \mathrm{im}(\mathrm{id}_{D*} - p) \subseteq \ker p\\ &\qquad\qquad\qquad\qquad \text{from Proposition 2.2.13)}\\ &= x\end{aligned}$$

(c) $(\mathrm{id}_{D*} - p)h = 0_{\mathrm{Hom}(D*, \ker p)}$;

$$\begin{aligned}(\mathrm{id}_{D*} - p)h &= (\mathrm{id}_{D*} - d_{D*}h - hd_{D*})h && \text{(by unfolding the definition of } p)\\ &= h - d_{D*}hh - hd_{D*}h\\ &= h - h && \text{(because } hh = 0_{\mathrm{End}(D*)} \text{ and}\\ &&& \text{also the premise } hd_{D*}h = h)\\ &= 0_{\mathrm{End}(D*)} && \text{(since } h \in \mathrm{End}(D*))\end{aligned}$$

In addition to this, it must be proved that $\mathrm{im}(\mathrm{id}_{D*} - p) \subseteq \ker p$, which follows from Proposition 2.2.13.

(d) $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p, D*)}$; let x be an element of $\ker p$, then,

$$\begin{aligned}h\,\mathrm{inc}_{\ker p}(x) &= h(x)\\ &= hd_{D*}h(x) && \text{(introducing the premise } hd_{D*}h = h)\\ &= hd_{D*}h(x) + hhd_{D*}(x) && \text{(since } hh = 0_{\mathrm{End}(D*)})\\ &= h(d_{D*}h + hd_{D*})(x)\\ &= h(p(x)) && \text{(introducing the definition of } p)\\ &= h0_{D*} && \text{(since } x \in \ker p)\\ &= 0_{D*}\end{aligned}$$

Finally we get that $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p, D*)}$.

(e) $hh = 0_{\mathrm{End}(D*)}$; this is one of the premises.

■

The previous lemma will be now used to build a new reduction between two chain complexes, once the perturbation has been introduced in the problem; this corresponds to the third lemma in our collection.

Lemma 2.2.15. *Under the BPL hypotheses,* and assuming the definitions and results given in Section 2.2.1, *there exists a canonical and explicit reduction* $(D'_*, d_{D'_*}) \Rightarrow (\ker p', d_{D'_*})$, *where* $p' = d_{D'_*}h' + h'd_{D'_*}$, *which is given through* $(\mathrm{id}_{D_*} - p', \mathrm{inc}_{\ker p'}, h')$.

Proof. First we recall the definitions given in the statement of the BPL (see Lemma 2.1.5) of $h' = h\phi$, $d_{D'_*} = d_{D_*} + \delta_{D_*}$, where δ_{D_*} is a perturbation of the differential d_{D_*}, and also the identities $\psi h = h\phi$ (Lemma 2.2.3), $\delta_{D_*}\psi = \phi\delta_{D_*}$ (Lemma 2.2.4), $\psi = \mathrm{id}_{D_*} - h\delta_{D_*}\psi = \mathrm{id}_{D_*} - \psi h\delta_{D_*} = \mathrm{id}_{D_*} - h\phi\delta_{D_*}$ (Lemma 2.2.5), and $\phi = \mathrm{id}_{D_*} - \delta_{D_*}h\phi = \mathrm{id}_{D_*} - \phi\delta_{D_*}h = \mathrm{id}_{D_*} - \delta_{D_*}\psi h$ (Lemma 2.2.6).

From the BPL hypotheses, there is a reduction (f, g, h) from (D_*, d_{D_*}) to (C_*, d_{C_*}), which means that $hh = 0_{\mathrm{End}(D_*)}$ and also that $hd_{D_*}h = h$ (see the Proof of Lemma 2.2.14).

In the following, we define $(D'_*, d_{D'_*})$ as $(D_*, d_{D_*} + \delta_{D_*})$, from which follows that $D'_* = D_*$ (this notation will allow us to distinguish between $d_{D'_*}$ and d_{D_*}).

In order to apply Lemma 2.2.14, three conditions must be satisfied:

1. The differential structure $(D'_*, d_{D'_*})$ must be a chain complex. From the premises it is clear that D'_* is a graded group. The condition for $d_{D'_*} = d_{D_*} + \delta_{D_*}$ of being a differential for D'_* follows from Definition 2.1.2 of perturbation.

2. The endomorphism h' of D'_* must satisfy the condition $h'h' = 0_{\mathrm{End}(D'_*)}$, which can be proved as follows:

$$\begin{aligned} h'h' &= \psi hh\phi && \text{(from the definition of } h' \text{ and also Lemma 2.2.3)} \\ &= 0_{\mathrm{End}(D_*)} && \text{(since } hh = 0_{\mathrm{End}(D_*)}\text{)} \\ &= 0_{\mathrm{End}(D'_*)} && \text{(as defined, } D'_* = D_*\text{)} \end{aligned}$$

   The degree of h' is $+1$, taking into account that the degree of h is $+1$ and both ϕ and ψ have degree 0 as established in Proposition 2.1.4.

3. The homotopy operator h' and the differential of the chain complex $(D'_*, d_{D'_*})$ must satisfy the property $h'd_{D'_*}h' = h'$:

$$\begin{aligned} h'd_{D'_*}h' &= \psi h(d_{D_*} + \delta_{D_*})h\phi && \text{(from definition of } d_{D'_*}, h' \text{ and Lemma 2.2.3)} \\ &= \psi hd_{D_*}h\phi + \psi h\delta_{D_*}h\phi && \text{(by distributivity)} \\ &= \psi h\phi + \psi h(\mathrm{id}_{D_*} - \phi) && \text{(due to Lemma 2.2.6 and } hd_{D_*}h = h\text{)} \\ &= \psi h && \text{(simplifying the obtained expression)} \\ &= h' && \text{(from the definition of } h'\text{)} \end{aligned}$$

Now, by introducing Lemma 2.2.14, we conclude that p' is a projector and the triple $(\mathrm{id}_{D'_*} - p', \mathrm{inc}_{\ker p'}, h')$ defines a reduction between the chain complexes $(D'_*, d_{D'_*})$ and $(\ker p', d_{D'_*})$. ■

The following lemma (the fourth one in our collection of six lemmas to obtain the complete proof of the BPL) explains how to build an isomorphism between the graded groups $\ker p$ and $\ker p'$; some previous properties about h', $d_{D'_*}$, h and d_{D_*} are proved before.

Proposition 2.2.16. *Under the BPL hypotheses, and with $\pi = \mathrm{id}_{D_*} - p$ (as defined in Lemma 2.2.11) and $\pi' = \mathrm{id}_{D_*} - p'$ (with $p' = d_{D'_*}h' + h'd_{D'_*}$, as defined in Lemma 2.2.15), the following properties hold:*

1. $h\pi' = 0_{\mathrm{End}(D_*)}$ *(of degree +1).*
2. $\pi' h = 0_{\mathrm{End}(D_*)}$ *(of degree +1).*
3. $\pi h' = 0_{\mathrm{End}(D_*)}$ *(of degree +1).*
4. $h'\pi = 0_{\mathrm{End}(D_*)}$ *(of degree +1).*

Proof. We assume the definitions given in the statement of the BPL (see Theorem 2.1.5) of $h' = h\phi$, $d_{D'_*} = d_{D_*} + \delta_{D_*}$ and also the equalities $\psi h = h\phi$ (Lemma 2.2.3), $\delta_{D_*}\psi = \phi\delta_{D_*}$ (Lemma 2.2.4), $\psi = \mathrm{id}_{D_*} - h\delta_{D_*}\psi = \mathrm{id}_{D_*} - \psi h\delta_{D_*} = \mathrm{id}_{D_*} - h\phi\delta_{D_*}$ (Lemma 2.2.5), and $\phi = \mathrm{id}_{D_*} - \delta_{D_*}h\phi = \mathrm{id}_{D_*} - \phi\delta_{D_*}h = \mathrm{id}_{D_*} - \delta_{D_*}\psi h$ (Lemma 2.2.6). Taking into account that the degree of both h and h' is +1, and also that the degree of π and π' is 0, it is clear that the degree of the compositions in the statement is equal to +1.

1. First we prove that $h\pi' = 0_{\mathrm{End}(D_*)}$;

$$\begin{aligned}
h\pi' &= h(\mathrm{id}_{D_*} - (d_{D'_*}h' + h'd_{D'_*})) && \text{(from the definition of } p'\text{)}\\
&= h - h(d_{D_*} + \delta_{D_*})h\phi && \text{(because } h' = h\phi,\ hh = 0_{\mathrm{End}(D_*)},\\
& && \text{and } d_{D'_*} = d_{D_*} + \delta_{D_*}\text{)}\\
&= h - hd_{D_*}h\phi - h\delta_{D_*}h\phi\\
&= h - h\phi - h(\mathrm{id}_{D_*} - \phi) && \text{(from } hd_{D_*}h = h \text{ and Lemma 2.2.6)}\\
&= 0_{\mathrm{End}(D_*)}
\end{aligned}$$

2. Now $\pi' h = 0_{\mathrm{End}(D_*)}$;

$$\begin{aligned}
\pi' h &= (\mathrm{id}_{D_*} - (d_{D'_*}h' + h'd_{D'_*}))h\\
&= h - \psi h(d_{D_*} + \delta_{D_*})h && \text{(since } h' = \psi h,\ hh = 0_{\mathrm{End}(D_*)},\\
& && \text{and } d_{D'_*} = d_{D_*} + \delta_{D_*}\text{)}\\
&= h - \psi hd_{D_*}h - \psi h\delta_{D_*}h\\
&= h - \psi h - (\mathrm{id}_{D_*} - \psi)h && \text{(from } hd_{D_*}h = h \text{ and Lemma 2.2.5)}\\
&= 0_{\mathrm{End}(D_*)}
\end{aligned}$$

3. In a similar way it can be proved that $\pi h' = 0_{\mathrm{End}(D_*)}$;

$$\begin{aligned}
\pi h' &= (\mathrm{id}_{D_*} - (d_{D_*} h + h d_{D_*})) h\phi && \text{(since } h' = h\phi) \\
&= (h - (d_{D_*} hh + h d_{D_*} h))\phi \\
&= (h - h)\phi && (hh = 0_{\mathrm{End}(D_*)} \text{ and} \\
& && \text{from the property } h d_{D_*} h = h) \\
&= 0_{\mathrm{End}(D_*)}
\end{aligned}$$

4. Finally, the fourth case, $h'\pi = 0_{\mathrm{End}(D_*)}$ can be obtained as follows:

$$\begin{aligned}
h'\pi &= \psi h(\mathrm{id}_{D_*} - (d_{D_*} h + h d_{D_*})) && \text{(since } h' = \psi h) \\
&= \psi(h - h d_{D_*} h - hh d_{D_*}) \\
&= \psi(h - h) && (hh = 0_{\mathrm{End}(D_*)} \text{ and} \\
& && \text{from the property } h d_{D_*} h = h) \\
&= 0_{\mathrm{End}(D_*)}
\end{aligned}$$

■

Lemma 2.2.17. *Under the BPL hypotheses,* and assuming the definitions and results given in Section 2.2.1*, there exists a canonical and explicit isomorphism between* $\ker p$ *and* $\ker p'$ as graded groups*, where* $p = d_{D_*} h + h d_{D_*}$ *and* $p' = d_{D'_*} h' + h' d_{D'_*}$.

Proof. We use again the definitions $\pi = \mathrm{id}_{D_*} - p$ and $\pi' = \mathrm{id}_{D_*} - p'$. In order to define the isomorphism between $\ker p$ and $\ker p'$, we will first prove the equalities between sets $\mathrm{im}\,\pi = \ker p$ and $\mathrm{im}\,\pi' = \ker p'$; then we will define a graded group isomorphism

$$\mathrm{im}\,\pi \cong \mathrm{im}\,\pi'$$

which finally will give us the way to build the isomorphism between $\ker p$ and $\ker p'$.

First, we prove that $\mathrm{im}\,\pi = \ker p$. We will see that they are mutual subsets.

1. We consider first the case $\mathrm{im}\,\pi \subseteq \ker p$. Let x be an element of $\mathrm{im}\,\pi$; we have to

verify that x belongs to $\ker p$, that is, $p(x) = 0_{D*}$. This can be proved as follows:

$$\begin{aligned}
p(x) &= p\pi(y) && \text{(with } y \in D*\text{, because } x \in \operatorname{im}\pi\text{)}\\
&= p(p\pi(y) + \pi\pi(y)) && \text{(from } \mathrm{id}_{D*} = \pi + p\text{,}\\
& && \text{follows that } \pi(y) = \pi\pi(y) + p\pi(y)\text{)}\\
&= pp\pi(y) + p\pi(y) && \text{(since } \pi \text{ is a projector,}\\
& && \text{as proved in Lemma 2.2.11)}\\
&= p\pi(y) + p\pi(y) && \text{(since } p \text{ is a projector,}\\
& && \text{as proved in Lemma 2.2.14)}\\
&= p(\mathrm{id}_{D*} - p)(y) + p(\mathrm{id}_{D*} - p)(y) && \text{(unfolding the definition of } \pi\text{)}\\
&= (p - pp)(y) + (p - pp)(y) && \text{(by distributivity)}\\
&= 0_{D*} && \text{(since } p \text{ is a projector,}\\
& && \text{as proved in Lemma 2.2.14)}
\end{aligned}$$

In the first step of the proof, one can be observe again a non-constructive argument, when the preimage of x is selected.

2. Now we give a proof of $\ker p \subseteq \operatorname{im}\pi$. Let x be an element in $\ker p$; x will be in the set $\operatorname{im}\pi$ if there exists an element $y \in D*$ such that $x = \pi(y)$:

$$\begin{aligned}
x &= p(x) + \pi(x) && \text{(unfolding the definition of } \pi\text{)}\\
&= \pi(x) && \text{(since } x \text{ is in } \ker p\text{, } p(x) = 0_{D*}\text{)}
\end{aligned}$$

Recalling that $\operatorname{im}\pi$ is a graded group, as showed in Proposition 2.2.10, and so is $\ker p$, proved in Proposition 2.2.9, $\mathrm{id}\colon \operatorname{im}\pi \to \ker p$ can be lifted to an isomorphism between both graded groups.

Secondly, we prove that $\operatorname{im}\pi' = \ker p'$, following a similar approach.

First, let us observe that π' satisfies a similar property as π, being also a projector:

$$\begin{aligned}
\pi'\pi' &= (\mathrm{id}_{D*} - p')(\mathrm{id}_{D*} - p') && \text{(unfolding the definition of } \pi'\text{)}\\
&= \mathrm{id}_{D*} - p' - p' + p' && \text{(since } p' \text{ is a projector)}\\
&= \pi'
\end{aligned}$$

Now, we prove that both sets $\operatorname{im}\pi'$ and $\ker p'$ are equal by proving that both of them are mutually contained.

1. In order to prove that $\operatorname{im}\pi' \subseteq \ker p'$, we consider $x \in \operatorname{im}\pi'$; x will be an element of $\ker p'$ if it satisfies that $p'x = 0_{D'_*}$:

$$\begin{aligned}
x &= \pi'(x) + p'(x) && \text{(unfolding the definition } \pi' = \mathrm{id}_{D'_*} - p'\text{)}\\
&= \pi'\pi'(y) + p'(x) && \text{(from } x \in \operatorname{im}\pi'\text{, } x = \pi'(y)\text{)}\\
&= \pi'(y) + p'(x) && \text{(since } \pi' \text{ is a projector)}\\
&= x + p'(x) && \text{(from } x = \pi'(y)\text{)}
\end{aligned}$$

Therefore, we obtain that $p'(x) = 0_{D'_*}$ and then it is an element of $\ker p'$. In the second step of the proof, we have used again a non-constructive argument involving the set $\operatorname{im} \pi'$.

2. Now we consider the case $\ker p' \subseteq \operatorname{im} \pi'$. Let x be an element of $\ker p'$; x will be in $\operatorname{im} \pi'$ if there exists an element $y \in D_*$ such that $x = \pi'(y)$:

$$\begin{aligned} x &= p'(x) + \pi'(x) && \text{(unfolding the definition } \pi' = \mathrm{id}_{D'_*} - p'\text{)} \\ &= \pi'(x) && \text{(since } x \text{ in } \ker p',\ p'(x) = 0_{D'_*}\text{)} \end{aligned}$$

Recalling that $\operatorname{im} \pi'$ is a graded group from Proposition 2.2.10 and so is $\ker p'$ from Proposition 2.2.9, $\mathrm{id}\colon \operatorname{im} \pi' \to \ker p'$ can be lifted to an isomorphism between both graded groups.

Now that we have proved that $\ker p$ and $\operatorname{im} \pi$ are isomorphic, and so are $\ker p'$ and $\operatorname{im} \pi'$, we can complete the proof by defining the homomorphisms τ and τ^{-1} between $\operatorname{im} \pi$ and $\operatorname{im} \pi'$.

Let us consider

$$\tau\colon \operatorname{im} \pi \to \operatorname{im} \pi'$$

given by $x \mapsto \pi'(x)$ and also

$$\tau^{-1}\colon \operatorname{im} \pi' \to \operatorname{im} \pi$$

mapping $x \mapsto \pi(x)$.

It can be seen that τ and τ^{-1} are mutually inverse.

1. To prove that $\tau\tau^{-1} = \mathrm{id}_{\operatorname{im} \pi'}$, let x be an element of $\operatorname{im} \pi'$:

$$\begin{aligned} \tau\tau^{-1}(x) &= \tau\tau^{-1}(\pi'(y)) && \text{(since } x \in \operatorname{im} \pi',\ x = \pi'(y)\text{)} \\ &= \pi'\pi\pi'(y) && \text{(unfolding the definitions of } \tau \text{ and } \tau^{-1}\text{)} \\ &= \pi'(\mathrm{id}_{D_*} - p)\pi'(y) && \text{(unfolding the definition of } \pi\text{)} \\ &= \pi'\pi'(y) - \pi' p \pi'(y) && \text{(by distributivity)} \\ &= \pi'(y) - \pi' p \pi'(y) && \text{(since } \pi' \text{ is a projector)} \\ &= x - \pi' p \pi'(y) && \text{(from } x = \pi'(y)\text{)} \\ &= x - \pi'(d_{D_*} h + h d_{D_*})\pi'(y) && \text{(unfolding the definition of } p\text{)} \\ &= x && \text{(from the properties of } \pi' \text{ obtained in} \\ & && \text{Proposition 2.2.16)} \end{aligned}$$

In the first step of the proof, we do not undertake the task of precisely constructing y; we just assume its existence from the definition of the image set.

2. To prove that $\tau^{-1}\tau = \mathrm{id}_{\mathrm{im}\,\pi}$, given $x \in \mathrm{im}\,\pi$,

$$\begin{aligned}
\tau^{-1}\tau(x) &= \tau^{-1}\tau(\pi(y)) && \text{(since } x \in \mathrm{im}\,\pi\text{)}\\
&= \pi\pi'\pi(y) && \text{(unfolding definitions of } \tau \text{ and } \tau^{-1}\text{)}\\
&= \pi(\mathrm{id}_{D_*} - p')\pi(y) && \text{(unfolding the definition of } \pi'\text{)}\\
&= \pi\pi(y) - \pi(d_{D'_*}h' + h'd_{D'_*})\pi(y) && \text{(unfolding the definition of } p'\text{)}\\
&= x - \pi(d_{D'_*}h' + h'd_{D'_*})\pi(y) && \text{(since } \pi \text{ is a projector and } \pi(y) = x\text{)}\\
&= x && \text{(from the properties of } \pi \text{ obtained in Proposition 2.2.16)}
\end{aligned}$$

Therefore, there is an isomorphism between the graded groups $\mathrm{im}\,\pi$ and $\mathrm{im}\,\pi'$, and considering the already known graded group isomorphisms between $\ker p$ and $\mathrm{im}\,\pi$ and also between $\ker p'$ and $\mathrm{im}\,\pi'$, we can define a graded group isomorphism between $\ker p$ and $\ker p'$ by composition, through τ and τ^{-1}. ■

The following lemma corresponds to the fifth one in our development of the BPL. In the previous one we have obtained an isomorphism between one of the graded groups appearing in the reduction given in the statement of the BPL, $\ker p$, and a graded group obtained after introducing the perturbation δ_{D_*}, $\ker p'$. Introducing also the reduction proposed in Lemma 2.2.14 from (D_*, d_{D_*}) to $(\ker p, d_{D_*})$, an isomorphism can be obtained between the graded groups C_* and $\ker p$. In the fifth lemma of the collection (Lemma 2.2.18) we define a way to transfer differentials from a chain complex to a graded group, once that an isomorphism between the underlying graded groups is known, and thus allowing us to establish relationships between chain complexes.

Lemma 2.2.18. *Let (A_*, d_{A_*}) be a chain complex, B_* be a graded group, and $F\colon A_* \to B_*$ (degree 0), $F^{-1}\colon B_* \to A_*$ (degree 0) define an isomorphism* between the graded groups A_* and B_*. *Then, the graded group homomorphism (degree -1) d_{B_*} defined by $Fd_{A_*}F^{-1}$ is a differential for the graded group B_* such that F and F^{-1} become inverse isomorphisms* between chain complexes.

Proof. First we prove that $d_{B_*} = Fd_{A_*}F^{-1}$ is a differential for the graded group B_*; following Definition 1.1.5 of chain complex, it must satisfy:

1. The differential $d_{B_*}\colon B_* \to B_*$ must be a graded group endomorphism (degree -1), which holds because F, F^{-1} are homomorphisms between graded groups of degree 0, and d_{A_*} is a graded group homomorphism of degree -1. The degree of the composition is -1 because F and F^{-1} have degree 0, and d_{A_*} degree -1.

2. The map must satisfy the property $d_{B_*}d_{B_*} = 0_{\mathrm{End}(B_*)}$.

$$\begin{aligned}
d_{B_*}d_{B_*} &= Fd_{A_*}F^{-1}Fd_{A_*}F^{-1}\\
&= Fd_{A_*}d_{A_*}F^{-1}\\
&= 0_{\mathrm{End}(B_*)}
\end{aligned}$$

In the second part it must be proved that F and F^{-1} are chain complex homomorphisms; they are graded group homomorphisms mutually inverse from the premises, and they should also be coherent with the differentials, as showed by $Fd_{A*} = Fd_{A*}(F^{-1}F) = d_{B*}F$, and using the same property $d_{A*}F^{-1} = (F^{-1}F)d_{A*}F^{-1} = F^{-1}d_{B*}$, as required. ∎

In the sixth lemma of the collection, we prove a property which explicitly builds a reduction between chain complexes provided that we have a previous reduction between two chain complexes as well as an isomorphism between one of these chain complexes and a third one:

Lemma 2.2.19. *Let $(f, g, h)\colon (A_*, d_{A_*}) \Rightarrow (B_*, d_{B_*})$ be a reduction and $F\colon (B_*, d_{B_*}) \to (C_*, d_{C_*})$ (degree 0) a chain complex isomorphism (being F^{-1} its inverse, with degree 0). Then $(Ff, gF^{-1}, h)\colon (A_*, d_{A_*}) \Rightarrow (C_*, d_{C_*})$ defines a reduction.*

Proof. From Definition 2.1.1 of reduction, the following conditions must be proved:

1. Ff and gF^{-1} must be chain complex homomorphisms (degree 0) between (A_*, d_{A_*}) and (C_*, d_{C_*}), which is satisfied since F, F^{-1}, f and g are chain complex homomorphisms (of degree 0).
2. h must be a homotopy operator (degree +1) for the chain complex (A_*, d_{A_*}), which follows from the premises.
3. The following relations must be satisfied:
   (a) $(Ff)(gF^{-1}) = \mathrm{id}_{C*}$; taking into account that $fg = \mathrm{id}_{A*}$ due to the properties of the reduction $(f, g, h)\colon (A_*, d_{A_*}) \Rightarrow (B_*, d_{B_*})$, the equality holds.
   (b) $(gF^{-1})(Ff) + d_{A*}h + hd_{A*} = \mathrm{id}_{A*}$; from the reduction properties of $(f, g, h)\colon (A_*, d_{A_*}) \Rightarrow (B_*, d_{B_*})$ follows that $(gf) + d_{A*}h + hd_{A*} = \mathrm{id}_{A*}$ and therefore the equality is satisfied.
   (c) $(Ff)h = 0_{\mathrm{Hom}(A*,C*)}$, which holds because f and h verify that $fh = 0_{\mathrm{Hom}(A*,B*)}$, from the reduction properties.
   (d) $h(gF^{-1}) = 0_{\mathrm{Hom}(C*,A*)}$, because h and g verify that $hg = 0_{\mathrm{Hom}(B*,A*)}$, from the reduction properties.
   (e) Finally, $hh = 0_{\mathrm{End}(A*)}$ follows from the reduction properties of (f, g, h).

∎

In Section 2.2.1 some lemmas stating properties about the series appearing in the statement of the BPL were introduced. In this section, using these properties, we have avoided the presence of the series. The last part of this section is devoted to produce a proof of the BPL using, exclusively, the six lemmas introduced in this section.

We recover now the original statement of the BPL:

Theorem 2.2.20. Basic Perturbation Lemma — *Let $(f, g, h)\colon (D_*, d_{D_*}) \Rightarrow (C_*, d_{C_*})$ be a chain complex reduction and $\delta_{D_*}\colon D_* \to D_*$ a* perturbation *of the differential d_{D_*} satisfying the nilpotency condition with respect to the reduction (f, g, h). Then a new reduction $(f', g', h')\colon (D'_*, d_{D'_*}) \Rightarrow (C'_*, d_{D'_*})$ can be obtained where the underlying graded groups D_* and D'_* (resp. C_* and C'_*) are the same, but the differentials are perturbed: $d_{D'_*} = d_{D_*} + \delta_{D_*}, d_{C'_*} = d_{C_*} + \delta_{C_*}$, and $\delta_{C_*} = f\delta_{D_*}\psi g$; $f' = f\phi$; $g' = \psi g$; $h' = h\phi$, where $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$, and $\psi = \sum_{i=0}^{\infty}(-1)^i(h\delta_{D_*})^i$.*

Proof. Both in the statement and in the proof of the lemma, $(D'_*, d_{D'_*})$ denotes the structure defined as $(D_*, d_{D_*} + \delta_{D_*})$. First, it can be observed that it is a chain complex since $(d_{D_*} + \delta_{D_*})$ is a differential for D_*, as follows from Definition 2.1.2 of perturbation applied to δ_{D_*}. Recall that considered as graded groups, both D'_* and D_* denote the same structure.

By applying Lemma 2.2.15 (the third one of the collection) a reduction:

$$(\mathrm{id}_{D'_*} - p', \mathrm{inc}_{\ker p'}, h')\colon (D'_*, d_{D'_*}) \Rightarrow (\ker p', d_{D'_*})$$

is obtained, where $h' = h\phi$, $p' = d_{D'_*}h' + h'd_{D'_*}$ and $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_{D_*}h)^i$.

Now by introducing Lemma 2.2.17 (the fourth one of the collection), an isomorphism between graded groups can be defined:

$$\pi\colon\ \ker p' \to \ker p$$

where π is the isomorphism gf, whose inverse was denoted by $\pi' = \mathrm{id}_{D'_*} - p'$ in Proposition 2.2.16.

A new isomorphism between chain complexes is defined by applying the second part of Lemma 2.2.11 (the first one of the collection):

$$f\colon (\mathrm{im}\, gf, d_{D_*}) \to (C_*, d_{C_*})$$

and from the proof of Lemma 2.2.17 (the fourth one), we already know that $\mathrm{im}\, gf = \ker p$, and therefore, an isomorphism can be defined:

$$f\colon (\ker p, d_{D_*}) \to (C_*, d_{C_*})$$

whose inverse is $g\colon (C_*, d_{C_*}) \to (\ker p, d_{D_*})$.

From the reduction $(\mathrm{id}_{D'_*} - p', \mathrm{inc}_{\ker p'}, h')\colon (D'_*, d_{D'_*}) \Rightarrow (\ker p', d_{D'_*})$ and the isomorphism between graded groups $\pi\colon\ \ker p' \to \ker p$, and applying Lemma 2.2.19 (the sixth one), a reduction can be defined:

$$(\pi(\mathrm{id}_{D'_*} - p'), \mathrm{inc}_{\ker p'}\pi', h')\colon (D'_*, d_{D'_*}) \Rightarrow (\ker p, \pi d_{D'_*}\pi')$$

Finally, by introducing Lemma 2.2.18 (the fifth one) with the chain complex $(\ker p, \pi d_{D'_*}\pi')$ and the graded group C_* and being f and g the isomorphisms between them, we obtain a new chain complex $(C_*, \pi d_{D'_*} \pi')$, and moreover, that f and g become now isomorphisms between these two chain complexes. Considering the reduction $(\pi(\mathrm{id}_{D'_*} - p'), \mathrm{inc}_{\ker p'} \pi', h')\colon (D'_*, d_{D'_*}) \Rightarrow (C_*, \pi d_{D'_*} \pi')$, the conditions of Lemma 2.2.19 (the sixth one) are satisfied, and by applying it we obtain a new reduction:

$$(f', g', h')\colon (D'_*, d_{D'_*}) \Rightarrow (C'_*, d_{C'_*})$$

Applying the formulas obtained in its proof, $D'_* = D_*$, $d_{D'_*} = d_{D_*} + \delta_{D_*}$, $C'_* = C_*$, $f' = f\pi(\mathrm{id}_{D'_*} - p')$, $g' = \mathrm{inc}_{\ker p'} \pi' g$, and the differential is $d_{C'_*} = f\pi(d_{D'_*})\pi' g$.

Some computations can be carried out in order to obtain simplified expressions for f', g' and $d_{C'_*}$. The value obtained for h' is $h\psi$, as announced in the statement.

First, g' can be simplified as follows:

$$\begin{aligned}
g' &= \mathrm{inc}_{\ker p'} \pi' g && \text{(from the obtained formula)}\\
&= \pi' g && \text{(because } \pi'\colon \ker p \to \ker p')\\
&= (\mathrm{id}_{D'_*} - p')g && \text{(unfolding the definition of } \pi')\\
&= (\mathrm{id}_{D'_*} - d_{D'_*} h' - h' d_{D'_*})g && \text{(unfolding the definition of } p')\\
&= g - d_{D'_*}\psi h g - \psi h d_{D_*} g - \psi h \delta_{D_*} g && \text{(from the obtained property } h' = \psi h)\\
&= g - \psi h d_{D_*} g - \psi h \delta_{D_*} g && \text{(from the reduction property, } hg = 0_{\mathrm{Hom}(C_*, D_*)})\\
&= -\psi h g d_{C_*} + (\mathrm{id}_{D'_*} - \psi h \delta_{D_*})g && \text{(using the differentials' property } d_{D_*} g = g d_{C_*})\\
&= (\mathrm{id}_{D'_*} - \psi h \delta_{D_*})g && \text{(from the reduction property, } hg = 0_{\mathrm{Hom}(C_*, D_*)})\\
&= \psi g && \text{(from the properties of Lemma 2.2.5)}
\end{aligned}$$

Now we simplify f':

$$\begin{aligned}
f' &= f\pi(\mathrm{id}_{D'_*} - p')g && \text{(from the obtained formula)}\\
&= f(\mathrm{id}_{D'_*} - d_{D'_*} h' + h' d_{D'_*}) && \text{(unfolding the definition of } p')\\
&= f - f d_{D_*} h' - f\delta_{D_*} h\phi - f h\phi d_{D'_*} && \text{(unfolding the definition of } h' = h\phi)\\
&= f - f d_{D_*} h' - f\delta_{D_*} h\phi && \text{(from the reduction property } fh = 0_{\mathrm{Hom}(C_*, D_*)})\\
&= f - d_{C_*} f h\phi - f\delta_{D_*} h\phi && \text{(using the differentials' property, } f d_{D_*} = d_{C_*} f)\\
&= f(\mathrm{id}_{D'_*} - \delta_{D_*} h\phi) && \text{(from the reduction property } fh = 0_{\mathrm{Hom}(C_*, D_*)})\\
&= f\phi && \text{(from the properties of Lemma 2.2.6)}
\end{aligned}$$

Finally, the differential $d_{C'_*}$ can be computed introducing Lemma 2.2.18. This lemma illustrates how the differential of a chain complex (in this case, $(\ker p, \pi d_{D'_*} \pi')$) has to be modified when its underlying graded group $(\ker p)$ is transformed through an isomorphism ($f\colon \ker p \to C_*$, $g\colon C_* \to \ker p$):

$$
\begin{aligned}
d_{C'_*} &= f\pi(d_{D'_*})\pi' g && \text{(introducing Lemma 2.2.18)}\\
&= fgf(d_{D*} + \delta_{D*})\pi' g && \text{(unfolding the definition of } \pi = gf)\\
&= f(d_{D*} + \delta_{D*})\pi' g && \text{(from the reduction property } fg = \mathrm{id}_{D*})\\
&= fd_{D*}\pi' g + f\delta_{D*}\psi g && \text{(using the property } \pi' g = \psi g\text{, obtained in the simplification of } g')\\
&= fd_{D*}(\mathrm{id}_{D'_*} - d_{D'_*}h' - h' d_{D'_*})g + f\delta_{D*}\psi g && \text{(unfolding the definition of } \pi')\\
&= fd_{D*}g - fd_{D*}d_{D'_*}h' g - fd_{D*}h' d_{D'_*}g + f\delta_{D*}\psi g && \text{(by associativity)}\\
&= fd_{D*}g + f\delta_{D*}\psi g && \text{(unfolding the definition of } h'\text{, the differentials properties and the reduction property } hg = 0_{\mathrm{End}(D*)})\\
&= d_{C*} + f\delta_{D*}\psi g && \text{(from the differentials property } d_{D*}g = gd_{C*}\text{, and the reduction property } fg = \mathrm{id}_{D*})
\end{aligned}
$$

The expressions in the statement of the BPL of f', g', h' and $d_{C'_*}$ have been established.

■

2.3 Ungraded version

In Chapter 3 we will explore the possibilities of a theorem prover to implement some fragments of the proof of the lemmas introduced in this chapter. With respect to the algebraic structures and the homomorphisms used in the previous lemmas, as graded groups and chain complexes, the information about the degree will not be considered. Consequently, graded groups will be substituted by abelian groups, and chain complexes by differential groups. This simplification does not affect to the validity of the lemmas and proofs, which still hold. Obviously, definitions must be modified in a coherent way (for instance, it is necessary to give an ungraded version of reduction), but it is a routine (trivial even) task.

For instance, the statement of the BPL with the modifications introduced would be as follows:

Theorem 2.3.1. Basic Perturbation Lemma (ungraded version) — *Let $(f, g, h)\colon (D, d_D) \Rightarrow (C, d_C)$ be a reduction between differential groups and $\delta_D\colon D \to D$ a* perturbation *of the differential d_D satisfying the nilpotency condition with respect to the reduction (f, g, h). Then a new reduction $(f', g', h')\colon (D', d_{D'}) \Rightarrow (C', d_{D'})$ can be obtained where the underlying abelian groups D and D' (resp. C and C') are the same, but the differentials are perturbed: $d_{D'} = d_D + \delta_D, d_{C'} = d_C + \delta_C$, and $\delta_C = f\delta_D\psi g$; $f' = f\phi$; $g' = \psi g$; $h' = h\phi$, where $\phi = \sum_{i=0}^{\infty}(-1)^i(\delta_D h)^i$, and $\psi = \sum_{i=0}^{\infty}(-1)^i(h\delta_D)^i$.*

The statements and proofs of the lemmas can be adapted in a similar way, obtaining the corresponding ungraded versions.

The reason why we will work without graduation is that the problems of implementing graded sets in Isabelle are completely independent of the central problems posed by the BPL proof. Thus, we have preferred to study first the essential points, assuming that the graded case requires only some additional technical resources in Isabelle. Even if enhancing these resources could be complicated, it is clear that the translation to that setting of the methods developed in the following chapter would be straightforward.

Chapter 3

Mechanizing the proof: a case study in Isabelle

3.1 Introduction

The object of formalization now will be the proof of the BPL which has been detailed in Section 2.2. In this chapter, we compare different approaches in which we have tried to implement fragments of the proof of the BPL in Isabelle. These approaches will be presented and also applied to some of the lemmas stated in Section 2.2.2. The question we are trying to address with these experiments is whether a complete implementation of the proof of the BPL can be done in Isabelle; the ideas introduced here are helpful to answer this question, as far as the fragments chosen are representative of the entire proof.

In Section 2.2, we emphasized that the purpose of giving a so detailed proof of the BPL, was to have a proof quite similar to the one we pretend to implement in a theorem prover. Therefore, the implementation of the proof should be also divided into two parts. In this chapter, we study in depth the problems posed by the lemmas presented in Section 2.2.2. The treatment of formal series as presented in Section 2.2.1 is left for future work and only briefly evoked in Chapter 5.

The structure of the chapter is as follows. In Section 3.2 the reasons that led us to use the theorem prover Isabelle will be presented. Then, in Section 3.3, we introduce a lemma and an overview of how the four different approaches that we have used in Isabelle can be applied to implement the proof of this lemma. Then, we briefly introduce in Section 3.4 a setting to study the representation of abstract (mathematical) objects and, in particular, of algebraic structures. This framework will be helpful to compare the different approaches. The four approaches will be described in detail in Sections 3.5, 3.6, 3.7 and 3.8. The differences among the approaches depend on the following aspects:

- Which of the algebraic structures appearing in the mathematical lemmas are implemented in the theorem prover. When we define in the theorem prover the context where the proofs are carried out, we can decide the algebraic structures which will be considered relevant for our task. The properties that we can apply in our proofs will depend on this decision.

- The implementation of homomorphisms between algebraic structures. Homomorphisms can be thought of, among other possibilities, as elements of a generic type α, but also as elements of a functional type $\alpha \Rightarrow \beta$. The first possibility improves the simplicity of proofs, but the second permits us to prove a wider range of lemmas.
- The implementation of the algebraic structures appearing in the lemmas. Algebraic structures can be represented just like a set, or like a set with operators satisfying some axioms. Depending on this design decision we will dispose of a different amount of previous knowledge that can be later applied to implement the proofs (associativity can not be directly applied to the elements of a magma; but if we prove a set to be a semigroup, then associativity will be ensured).

Each section will present the design decisions on these three aspects; then, the lemmas that can be proved will be enumerated, as well as the difficulties found, and the main reasons to introduce a new approach. We will also comment on the advantages and the drawbacks of each method, as well as the feasibility of applying them into different problems dealing with algebraic structures.

Part of the material of this chapter has been presented in [Aransay et al., 2002a, Aransay et al., 2002b, Aransay et al., 2002c, Aransay et al., 2003, Aransay et al., 2004].

3.2 The theorem prover: Isabelle

The first decision in the process of implementation of the proof of the BPL in a theorem prover is which theorem prover among the existing ones is the most appropriate for our task. A detailed comparison of the available theorem provers can be found in [Wiedijk, 2003]. In our case, we will not intend to analyze the features of the different theorem provers. We will focus our attention on the theorem prover Isabelle, and we will give an enumeration of the main reasons that led us to use it.

Isabelle/HOL [Nipkow et al., 2002] has some special features that will be useful for our purposes:

1. It has been written in the functional programming language ML; ML ([Paulson, 1996]) was originally designed to serve as the programming language for the theorem prover Edinburgh LCF (although it can be also used for other proposals; see, for instance `http://www.cs.princeton.edu/~appel/smlnj/projects.html`). Kenzo [Dousson et al., 1999] has been written in Common Lisp [Graham, 1996], which is a functional programming language even if, in

addition, it includes also object oriented features. Both tools, Isabelle and Kenzo, have been developed using functional programming languages. In addition to this, some Homological Algebra algorithms present in Kenzo have been already implemented in ML (see for instance [Andrés et al., 2003]). Moreover, HOL can be seen as an ML-like functional language. The main difference is that HOL functions are total (always terminate), while ML functions need not to terminate.

2. Isabelle contains an implementation of higher-order logic (Isabelle/HOL) in its standard distribution. Isabelle is a generic theorem prover, which means that it has a meta-logic, called Isabelle/Pure, and, on the top of it, a wide range of logics can be implemented (see the paper [Paulson, 1990b] for a detailed description of the Isabelle meta-logic). In the Isabelle distribution can be found, at least, implementations of first-order logic (FOL), higher-order logic (HOL) and Zermelo Fraenkel set theory (ZF). We have chosen higher-order logic for the implementation of the proofs we have proposed in Section 2.2.2, due to its expressiveness and to the previous works that have been developed with it in Group Theory.

3. Isabelle/HOL has been used to obtain formalized mathematics in different areas such as Number Theory, Analysis or Geometry. Therefore, libraries about Algebra and Group Theory have been already implemented in Isabelle/HOL and are available in the standard distribution; some of this work can be found in [Ballarin, 1999] and [Kammüller and Paulson, 1999]. These previous results show the advantages of using Isabelle/HOL in order to implement proofs in the domain of Abstract Algebra. Moreover, most of the concepts of Homological Algebra involved in our work (enumerated in Section 1.1) have been defined in terms of Group Theory structures, and then the existing libraries can be reused.

4. During the development of this work the Isar proof language was built on the top of Isabelle; Isar [Wenzel, 2004] is an extension of Isabelle which helps to obtain structured and human-readable proofs. The tactic style proofs in Isabelle were substituted for proofs in the Isar language that turned Isabelle/Isar[1] into a proof assistant system, as well as a theorem prover. The Isar language has been useful for the the development of our proofs, and is, undoubtedly, much easier to learn than the Isabelle tactic style. Moreover, proofs can be done, not only backwards (from the goals to the premises), as expected in the tactic style proofs, but also starting from the premises and building up the goal.

5. A program extraction facility is being implemented for Isabelle proofs (see [Berghofer, 2003a, Berghofer, 2003b, Berghofer, 2004]), which is capable of working with a constructive fragment of Isabelle/HOL. This tool helps us to obtain certified ML programs from our proofs (the results obtained related to this part of our work will be presented in Chapter 4).

[1]For the sake of readability, we will refer to Isabelle/Isar/HOL as Isabelle.

3.3 An introductory example

Once a tool for the implementation has been chosen, in a second stage the degree of formalization of the proofs is to be considered. This level of formalization will be illustrated with the following lemma. It can be also seen as an example of the kind of mathematical theorems we are trying to implement. This lemma corresponds to the ungraded version of Lemma 2.2.14, where we have substituted the differential groups by abelian groups, in order to focus our attention on the relevant parts of the proof, avoiding the technicalities derived of introducing differential structures. The main reason to choose this lemma is that it contains most of the interesting problems that can be found in the whole collection of lemmas in Section 2.2.2. It requires reasoning with homomorphisms and endomorphisms as if they were elements of certain algebraic structures, but also dealing with their functional definition; in addition to this, the domain conditions of the source or the target of the homomorphisms also play a role in some steps of the proof.

Lemma 3.3.1. *Let G be an abelian group and let h and d be elements of* $\mathrm{End}(G)$ *such that $hh = 0_{\mathrm{End}(G)}$, d is a differential for G and $hdh = h$; let us define $p = dh + hd$ and consider* $\ker p$*, abelian subgroup of G. Then, the following equalities hold (i.e., the triple* $(\mathrm{id}_{\mathrm{End}(G)} - p, \mathrm{inc}_{\ker p}, h)$ *defines a reduction from G to* $\ker p$*):*

1. $(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\mathrm{End}(\ker p)}$;
2. $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(G)} - p) + dh + hd = \mathrm{id}_{\mathrm{End}(G)}$;
3. $(\mathrm{id}_{\mathrm{End}(G)} - p)h = 0_{\mathrm{Hom}(G,\ker p)}$;
4. $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p,G)}$;
5. $hh = 0_{\mathrm{End}(G)}$.

Proof. The proof is divided into the five properties to be checked:

1. Let x be an element of $\ker p$:

$$\begin{aligned}(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p}(x) &= (\mathrm{id}_{\mathrm{End}(G)} - p)(x) && \text{(from the definition of the inclusion)}\\ &= x - p(x)\\ &= x && \text{(from the definition of the kernel set)}\end{aligned}$$

   Thus, $(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\mathrm{End}(\ker p)}$.

2. First we prove that p is a projector:

$$\begin{aligned}pp &= (dh + hd)(dh + hd) && \text{(unfolding the definition of } p\text{)}\\ &= dhdh + dhhd + hddh + hdhd\\ &= dh + hd && \text{(from } hdh = h,\ hh = 0_{\mathrm{End}(G)},\\ & && \text{and } dd = 0_{\mathrm{End}(G)}\text{)}\\ &= p\end{aligned}$$

Now let x be in G; in order the expression $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(G)} - p)(x)$ to be well-defined, it must be checked that $(\mathrm{id}_{\mathrm{End}(G)} - p)(x) \in \ker p$:

$$\begin{aligned} p(\mathrm{id}_{\mathrm{End}(G)} - p)(x) &= p(x) - pp(x) \\ &= p(x) - p(x) \qquad \text{(since } p \text{ is a projector)} \\ &= 0_G \end{aligned}$$

Then, the property can be proved as follows

$$\begin{aligned} (\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}\,G} - p) + dh + hd)(x) &= \mathrm{inc}_{\ker p}(x - p(x)) + (dh + hd)(x) \\ &= x - p(x) + (dh + hd)(x) \quad \text{(using that } x - p(x) \in \ker p) \\ &= x - p(x) + p(x) \quad \text{(unfolding the definition of } p) \\ &= x \end{aligned}$$

Therefore, we have proved $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(G)} - p) + dh + hd = \mathrm{id}_{\mathrm{End}(G)}$.

3. First, the following computation is carried out:

$$\begin{aligned} (\mathrm{id}_{\mathrm{End}(G)} - p)h &= h - ph \\ &= h - (dh + hd)h \qquad \text{(unfolding the definition of } p) \\ &= h - hdh \qquad \text{(using that } hh = 0_{\mathrm{End}(G)}) \\ &= h - h \qquad \text{(introducing that } hdh = h) \\ &= 0_{\mathrm{Hom}(G, \ker p)} \end{aligned}$$

Thus, we obtain that $(\mathrm{id}_{\mathrm{End}(G)} - p)h = 0_{\mathrm{Hom}(G, \ker p)}$.

4. Let x be an element in $\ker p$:

$$\begin{aligned} h\,\mathrm{inc}_{\ker p}(x) &= h(x) \\ &= hdh(x) \qquad \text{(from the property } hdh = h) \\ &= hdh(x) + hhd(x) \qquad \text{(from the property } hh = 0_{\mathrm{End}(G)}) \\ &= h(dh + hd)(x) \\ &= hp(x) \qquad \text{(from the definition of } p) \\ &= 0_{\ker p} \end{aligned}$$

Therefore, $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p, G)}$.

5. $hh = 0_{\mathrm{End}(G)}$ was one of the premises.

■

The four different approaches that we will further explain in detail in Sections 3.5, 3.6, 3.7 and 3.8, can be applied to implement the proof of Lemma 3.3.1. The different approaches will be applied in this example to the mathematical proof of the lemma. In the detailed description of the approaches we will also comment on, first, the implementation of the objects appearing in each of the different approaches, and second, on the implementation of the proofs in these approaches, in the theorem prover Isabelle.

The following mathematical structures have been used (explicitly or implicitly) in the mathematical proof of Lemma 3.3.1:

- The abelian group G and its subgroup $\ker p$.
- The ring of endomorphisms $\mathrm{End}(G)$.
- The ring of endomorphisms $\mathrm{End}(\ker p)$.
- The abelian groups of homomorphisms $\mathrm{Hom}(\ker p, G)$ and $\mathrm{Hom}(G, \ker p)$.
- An algebraic structure for homomorphisms allowing to operate, for instance, elements of $\mathrm{Hom}(\ker p, G)$ with elements of $\mathrm{End}(G)$ or elements of $\mathrm{Hom}(G, \ker p)$ with elements of $\mathrm{End}(\ker p)$ through the usual composition, $\circ$. This structure can be formalized as a ringoid (see Definition 1.1.24).

The differences between the four proposed approaches, as we have detailed in Section 3.1, will depend on three factors; first, the choice of the algebraic structures present in the lemma that are represented in the mechanized proof of the lemma, second, the representation of the chosen algebraic structures, and third, the representation of the homomorphisms between the algebraic structures.

Each of these points will be treated separately in the four given approaches. In this example, we first describe each of these three points in the different approaches, and then we implement the parts of the proof of Lemma 3.3.1 that can be carried out in each approach.

Let us consider the first and the fifth equality in the statement of the lemma. Their proofs can be implemented in the setting that we will expose in the *symbolic approach*, in Section 3.5. In the symbolic approach, the three aspects are as follows:

1. The only algebraic structure from the lemma that is represented is the ring $\mathrm{End}(G)$, through a record type which allows us to group the carrier an the operations of the algebraic structure together (see Section 1.3.2).

2. The homomorphisms (only endomorphisms in this case) are represented through elements of a generic type $'a$.

3. The ring of endomorphisms is represented through a generic ring R, whose elements, of a generic type $'a$, are interpreted as endomorphisms of G.

The endomorphisms d, h, p, $\mathrm{id}_{\mathrm{End}(G)}$ and $0_{\mathrm{End}(G)}$[2] of the abelian group G can be considered as elements of this generic ring R and then the equalities can be proved as follows:

- $hh = 0_R$ is one of the premises.
- $(\mathrm{id}_R - p)h = 0_R$ because $(\mathrm{id}_R - p)h = h - ph$ and from the premises $ph = h$.

For these two properties, representing $\mathrm{End}(G)$ through a generic ring R is enough. Information about the concrete representation of the endomorphisms has been neither provided, nor needed. This setting corresponds to our first approach, which was already presented in [Aransay et al., 2002c] and will be studied in Section 3.5.

On the contrary, this setting is not enough to prove some other equations in Lemma 3.3.1. For instance, no proof of the third property $(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\mathrm{End}(\ker p)}$ can be given inside of this framework; the elements $\mathrm{id}_{\mathrm{End}(G)}$ and p can be represented through id_R and also p in R, but no representation can be given neither for $\mathrm{inc}_{\ker p}$, nor for $\mathrm{id}_{\mathrm{End}(\ker p)}$, in the ring R. Our representation in Isabelle has omitted information which is needed for the statements to be meaningful.

Thus, a new approach must be proposed. In our first alternative, the three main features of the framework are:

1. We introduce now two algebraic structures, the abelian group G and also the abelian subgroup $\ker p$.
2. They will be represented like records of a generic type (we will give a more detailed explanation in Section 3.6).
3. Homomorphisms will be represented through elements of functional type $'a \Rightarrow 'b$ (taking into account that $\ker p$ is a subgroup of G, the type of homomorphisms between both structures will be a specialization of the previous one, $'a \Rightarrow 'a$); in contrast with the previous approach, the ring of endomorphisms will not be represented.

This representation corresponds to the existing one in the Isabelle libraries for Group Theory. Now, a proof for the third property of Lemma 3.3.1, which was not provable in the symbolic approach, stating that $(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\mathrm{End}(\ker p)}$ can be implemented following this argument. Let x be an element of $\ker p$, then

$$\begin{aligned}(\mathrm{id}_{\mathrm{End}(G)} - p)\,\mathrm{inc}_{\ker p}(x) &= (\mathrm{id}_{\mathrm{End}(G)} - p)(x) && (\text{since } x \in \ker p)\\ &= x - p(x) \\ &= x && (\text{because of the definition of the kernel set})\end{aligned}$$

[2]Actually, the homomorphism in the statement is $0_{\mathrm{Hom}(G,\ker p)}$, but, from a functional point of view, it can be identified, *in this context*, with the endomorphism $0_{\mathrm{End}(G)}$.

We have named this framework *set theoretic approach* due to two reasons; first, because it represents the algebraic structures collecting endomorphisms and homomorphisms through sets, and second, because homomorphisms are assigned a functional type, but in their axiomatic definition sets over these types will have to be considered. We will present it in detail in Section 3.6. The representation chosen in this framework seems to be enough to prove the rest of the properties (the second and the fourth) of reduction, at least from a theoretical point of view. On the other hand, when we tried to implement these proofs, some problems were detected slowing down the proofs performance and impoverishing the presentation. The main reason is the weak representation through sets of some important algebraic structures (the rings $\mathrm{End}(G)$ and $\mathrm{End}(\ker p)$ or the abelian groups $\mathrm{Hom}(G, \ker p)$ and $\mathrm{Hom}(\ker p, G)$ among others), instead of providing them with adequate operations. In addition, some problems derived of implementing homomorphisms through total functions over types emulating functions between sets arise. We will avoid the enumerated problems by improving the representation in Isabelle of some of these algebraic structures and changing the representation of the homomorphisms, as will be seen in our third approach, which we name *morphism based approach*. This approach was presented in [Aransay et al., 2003].

This third setting is exposed in Section 3.7, and owes its name to the fact that the representation of homomorphisms has been changed. Now, their representation is quite similar to the usual representation of morphisms in a Category Theory setting. The most important features are, accordingly to our list:

1. The algebraic structures present will be the same as in the set theoretic approach, the abelian groups G and $\ker p$, and, in addition to this, also the ring $\mathrm{End}(G)$ of the endomorphisms of G.

2. The algebraic structures are represented in the same way as in the previous approach, through extensible records. In order to obtain a ring with the endomorphisms of G, some more axioms have to be imposed to homomorphisms. These axioms ensure, for instance, the uniqueness of the representation of each homomorphism and the compatibility with the operators, allowing us then to prove in Isabelle the ring properties of the data structure representing $\mathrm{End}(G)$.

3. The homomorphisms will be modified in two senses with respect to the representation given in the previous set theoretic approach: the type and also the axiomatic characterization. First, they will be represented like records, which store information about their functional nature (with type $'a \Rightarrow 'b$), and also about their source and target algebraic structures (in a similar fashion as morphisms are usually represented in Category Theory; see the works in Isabelle [Glimming, 2001, O'Keefe, 2004]). This information will be used for the compositions of homomorphisms and endomorphisms, avoiding some partiality issues. In the axiomatic characterization, we will have to enrich the definition in the Isabelle library with axioms ensuring the uniqueness of the representation (i.e., a homomorphism in the mathematical setting will be represented just by one homomorphism in our Isabelle environment).

This framework also incorporates some new lemmas about composition of the defined homomorphisms, and about how to change in a secure way the source and the target of a homomorphism under determined circumstances. All the introduced tools are enough to work with the homomorphisms and the endomorphisms equationally, just deriving the properties from the algebraic structures they belong to, as in the symbolic approach, and also to dispose of the functional information on them, as in the set theoretic approach. We will further comment on the details; in a few words, it can be said that the advantages of the symbolic approach and the set theoretic approach have been gathered in this morphism based approach.

Finally a fourth approach is introduced in Section 3.8. Using an Isabelle tool, interpretation of locales, which has been recently implemented, we can build the following framework:

1. The algebraic structures present will be the same as in the morphism based approach: the abelian groups G and $\ker p$, the ring of endomorphisms of G, and, in addition, we also introduce the ring of endomorphisms $\operatorname{End}(\ker p)$, and the abelian groups $\operatorname{Hom}(G, \ker p)$ and $\operatorname{Hom}(\ker p, G)$.

2. The representation of homomorphisms lies on the ideas given for the morphism based approach; each homomorphism must have a unique representation, and therefore the axiomatic part is quite similar. The only difference is that, now, it will be enough to keep the functional part of homomorphisms, without storing explicitly their source or target.

3. The representation of the algebraic structures G and $\ker p$ is the same as in the previous approach, through extensible records. The rings $\operatorname{End}(G)$ and $\operatorname{End}(\ker p)$, as well as the abelian groups $\operatorname{Hom}(\ker p, G)$ and $\operatorname{Hom}(G, \ker p)$ will be also represented through records. The main difference with respect to the morphism approach is that, now, we have explicitly represented the algebraic structures $\operatorname{End}(G)$, $\operatorname{End}(\ker p)$, $\operatorname{Hom}(\ker p, G)$ and $\operatorname{Hom}(G, \ker p)$, and consequently, we can take advantage of the properties of these algebraic structures in the proofs.

Locales are helpful to encode all these structures in such a way that we can later recover (in a neat way) the operations and properties of each of them (even when they do not increase the reasoning power of Isabelle). We will introduce this framework in Section 3.8. It has been named *interpreting locales approach*, because of the technique we use to create a context where proofs are developed.

3.4 Encoding mathematics

Whenever we represent a mathematical problem in a computer, in a theorem prover or in a symbolic computation system (and, generally speaking, in any software or hardware system), a process of abstraction is carried out between the objects we are considering in our computer and the actual mathematical entities. Examples are obvious: in an ASCII encoding, the symbol “2” is coded through “0011 0010” in its binary form, whereas in a binary coded decimal (or BCD) system, like the one used in COBOL, the representation of “2” would be just “0010”. In other words, in the machine we can find a *representation* of the mathematical entities we are dealing with. When we represent algebraic structures or functions in a theorem prover, such a process of representation is also present. We do not pretend to undertake in this memoir an exhaustive study of this process, but we consider here some ideas about it which can clarify the differences among the approaches that will be later introduced, as well as the possibilities and limitations of each one of them.

We have already said that the main constituents of each framework are the algebraic structures and the homomorphisms connecting them. In this section, we propose a general framework allowing us to formally define what a *representation* of a mathematical entity means. We understand here for mathematical entities, at least, values, sets, algebraic structures and functions, although the definitions given could be used beyond these entities (see, for instance, [Pascual, 2002]).

The ideas given in this section are based on [Hoare, 1972, Loeckx et al., 1996], although we adopt here the notation introduced in [Pascual, 2002]. A representation $\mathcal{R}$ is formed by the following elements:

1. A collection $\mathcal{M}_{\mathcal{R}}$ containing the mathematical entities whose representation we pretend to achieve, named *model of the representation.*

2. A computer domain $\mathcal{D}_{\mathcal{R}}$, named *domain of the representation*, and which usually will be identified with a data type.

3. A collection of elements of $\mathcal{D}_{\mathcal{R}}$, named *carrier of the representation* and denoted by $\mathcal{S}_{\mathcal{R}}$.

4. A process of abstraction defined from the elements of $\mathcal{S}_{\mathcal{R}}$ into the elements of $\mathcal{M}_{\mathcal{R}}$, and named $\Lambda_{\mathcal{R}}$, which allows the user to identify the elements implemented with the mathematical entities they are representing (studying the properties of the process of abstraction, i.e., if it is a correspondence, a function, an injective function, a bijection, and so on, will provide us with relevant information about the representation).

5. An equivalence relation, $=_{\mathcal{R}}$, which identifies objects in the carrier of the representation, $\mathcal{S}_{\mathcal{R}}$, in such a way that whenever it is satisfied that $\mathtt{x} =_{\mathcal{R}} \mathtt{y}$, then the mathematical elements represented through `x` and `y` are equal in $\mathcal{M}_{\mathcal{R}}$.

We now illustrate the different elements of the process of representation with some examples in ML:

Example 3.4.1. A representation of the set of boolean values $Bool = \{True, False\}$ is $\mathcal{R}_{Bool} = (Bool, \mathtt{bool}, \mathtt{bool}, \Lambda_{Bool}, \mathtt{=})$, where `bool` has the values `true` or `false` and Λ_{Bool} is a function which maps `true` to *True* and `false` to *False.*

In this example, Λ_{Bool} is a bijective function between the elements of the *Bool* set and the ML objects with type `bool` (in particular, $\mathcal{S}_{Bool} = \mathcal{D}_{Bool}$). These representations, characterized by a bijective *total* function between the model and the carrier, are usually called *literal representations*.

Example 3.4.2. A new example appears when we try to represent a set of elements A in ML. We name this representation $\mathcal{R}_A$. In this case the model $\mathcal{M}_{\mathcal{R}_A}$ will be the set A. Then, we should choose an ML type to be the domain of the representation. As long as we are in a generic case, we choose an ML type variable, such as α, to represent this set. Thus, the domain of representation $\mathcal{D}_{\mathcal{R}_A}$ will be the type α. The carrier set $\mathcal{S}_{\mathcal{R}_A}$ will be determined by the concrete set A, and therefore we now leave it undetermined. The equality of the abstraction will be based on the boolean data type defined in ML, as presented in Example 3.4.1. The equality of the representation $=_{\mathcal{R}_A}$ will have type $\alpha \to \alpha \to$ `bool`, and will be identified with the equality in α.

Now we apply the previous example to some concrete sets.

Example 3.4.3. First, we pretend to achieve a representation of the set of natural numbers $\mathbb{N}$, that we will name $\mathcal{R}_{\mathbb{N}}$. The model $\mathcal{M}_{\mathcal{R}_{\mathbb{N}}}$ will be the set of natural numbers, $\mathbb{N}$. In ML there is a predeclared type `int`, which represents the set of integer numbers $\mathbb{Z}$, but there is no predeclared type for the natural numbers. Thus, we define the following representation. In this approach, we consider as domain $\mathcal{D}_{\mathcal{R}_{\mathbb{N}}}$ the ML type `int`, formed by the integer numbers, and which is known to be a type with equality. The carrier set of our representation will be formed just by the positive numbers, or, which is the same, the set $\mathcal{S}_{\mathcal{R}_{\mathbb{N}}} = \{$`0, 1, 2, ...`$\}$. The equality of the representation $=_{\mathcal{R}_{\mathbb{N}}}$ is defined to be the usual equality for integers in ML, =, restricted to the carrier. With this definition, the process of abstraction $\Lambda_{\mathcal{R}_{\mathbb{N}}}$ is a function such that assigns to every positive number `n` in ML the natural number n of the model $\mathbb{N}$.

It can be observed how the process of abstraction is a partial injective function from the domain set $\mathcal{D}_{R_{\mathbb{N}}}$ into the model of the representation $\mathbb{N}$. Whenever the abstraction function is an injective function, we will say that the representation is *faithful*.

Example 3.4.4. A new representation can be given for the natural numbers. In this second approach, that we will name $\mathcal{R}_{Nat}$, the model and the domain of the representation will be the same as in Example 3.4.3. Therefore, $\mathcal{M}_{\mathcal{R}_{Nat}}$ will be the set $\mathbb{N}$ and $\mathcal{D}_{\mathcal{R}_{Nat}}$ will be the ML type `int`. Now, as carrier set $\mathcal{S}_{\mathcal{R}_{Nat}}$ of the representation, we consider the whole domain of the representation, or, which is the same, all the elements with type `int`. The process of abstraction is defined as follows: for every element `n` with type `int` such that `n>=0`, we define $\Lambda_{\mathcal{R}_{Nat}}($`n`$) = n$, and for every `n` such that `n<0`, we define $\Lambda_{\mathcal{R}_{Nat}}($`n`$) = -n$.

The equality of the abstraction now must be changed, since the standard equality in ML is not coherent. The equality $=_{\mathcal{R}_{Nat}}$ that we consider now is given by `m` $=_{\mathcal{R}_{Nat}}$ `n` if `m = n` or `m = - n`.

It is worth noting some facts on this new representation with respect to the previous representation given in Example 3.4.3. First, the abstraction process $\Lambda_{\mathcal{R}_{Nat}}$ is now a total function from the domain of the representation into the model. Second, it is not injective, as far as every element n in $\mathbb{N}$, apart from 0, has two preimages `{n, -n}`. Finally, the equality of the abstraction $=_{\Lambda_{\mathcal{R}_{Nat}}}$ has been changed for the first time in our collection of examples, and it is not the standard equality in ML.

Example 3.4.5. We consider again the question of giving a representation for a set A, where A is now $\mathbb{Z}/5\mathbb{Z}$. The representation will be named $\mathcal{R}_{\mathbb{Z}/5\mathbb{Z}}$. The previous set can be expressed as $\{0, 1, 2, 3, 4\}$, which will be our model of the representation $\mathcal{M}_{\mathcal{R}_{\mathbb{Z}/5\mathbb{Z}}}$. Different representations can be given for this set, although we will consider in all our cases the domain of the representation as the ML type `int`:

1. We can consider as carrier set of the representation $\mathcal{S}_{\mathcal{R}_{\mathbb{Z}/5\mathbb{Z}}}$ the set of `int` elements $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}\}$. The abstraction process $\Lambda_{\mathcal{R}_{\mathbb{Z}/5\mathbb{Z}}}$ can be defined as the restriction of $\Lambda_{\mathcal{R}_{\mathbb{N}}}$ (see Example 3.4.3) to the elements of the carrier. Therefore, the abstraction process defines a bijective function from the carrier set into the model of the representation. The equality of the representation is defined as the ML equality restricted to the carrier.

2. A second representation can be given if we consider as the carrier set of the representation the data type `int`, and as abstraction function a function assigning to each term `n` of type `int`, the image through $\Lambda_{\mathcal{R}_{\mathbb{N}}}$ of `n mod 5`. Then, the abstraction function is no longer injective but it is still surjective, since every element of the model is represented through more than one element of the carrier. The equality of the representation would be the standard equality in ML restricted to the elements of the carrier (i.e., `n = m (mod 5)`).

Example 3.4.6. Now we give a representation for functions between mathematical sets. We consider the functions of the form $f\colon A \to B$ (i.e., the set of functions B^A). Two different problems must be faced. First, the representation of the mathematical sets A and B, and then, the representation of mathematical functions of the form $f\colon A \to B$.

The representation of generic sets will be made as exposed in Example 3.4.2. We consider two ML variable types, α for set A and β for set B, and both representations can be expressed as $\mathcal{R}_A = (A, \alpha, \mathcal{S}_{\mathcal{R}_A}, \Lambda_{\mathcal{R}_A}, =_{\mathcal{R}_A})$ and $\mathcal{R}_B = (B, \beta, \mathcal{S}_{\mathcal{R}_B}, \Lambda_{\mathcal{R}_B}, =_{\mathcal{R}_B})$, where $=_{\mathcal{R}_A}$ and $=_{\mathcal{R}_B}$ denote the equality of the representations.

Our representation $\mathcal{R}_{B^A}$ of the set of functions from A to B will be as follows. First, we choose as domain of the representation $\mathcal{D}_{\mathcal{R}_{B^A}}$ the ML type $\alpha \Rightarrow \beta$, where it can be observed that α is equal to $\mathcal{D}_{\mathcal{R}_A}$ and β is $\mathcal{D}_{\mathcal{R}_B}$; this ML type corresponds to terms which have an input of type α and return a value of type β.

The carrier set of the representation $\mathcal{S}_{\mathcal{R}_{B^A}}$ will be the elements of the set of ML functions $\{\mathtt{f} \mid \mathtt{f} \in \mathcal{S}_{\mathcal{R}_A} \to \mathcal{S}_{\mathcal{R}_B}\}$ such that whenever $\Lambda_{\mathcal{R}_A}(\mathtt{x}) = \Lambda_{\mathcal{R}_A}(\mathtt{y})$, then $\Lambda_{\mathcal{R}_B}(\mathtt{f\ (x)}) = \Lambda_{\mathcal{R}_B}(\mathtt{f\ (y)})$. From our perspective, it is enough to define the representation in this way. Nevertheless, there are intrinsical problems related to this representation that have been studied in depth, for instance, in [Pascual, 2002].

The abstraction process $\Lambda_{\mathcal{R}_{BA}}$ is such that, given an ML function `f` from the carrier of the representation, $\Lambda_{\mathcal{R}_{BA}}(\texttt{f})$ is a function f verifying that for all `x` in the carrier set $\mathcal{S}_{\mathcal{R}_A}$, such that $\Lambda_{\mathcal{R}_A}(\texttt{x}) = x$, with x in A, $(\Lambda_{\mathcal{R}_{BA}}(\texttt{f}))(\Lambda_{\mathcal{R}_A}(\texttt{x})) = f(x)$ (which thanks to the coherence among representations will be also equal to $(\Lambda_{\mathcal{R}_B})(\texttt{f}(\texttt{x}))$).

From the definition of the representation process $\Lambda_{\mathcal{R}_{BA}}$ it can be seen that several ML function (or none) could fulfill this condition for each function $f \in B^A$, and both injectivity and surjection of the abstraction function is not ensured.

The equality of the representation $=_{\mathcal{R}_{BA}}$ will be such that, given `g` and `h` in the carrier of the representation, it satisfies that $\texttt{g} =_{\Lambda_{\mathcal{R}_{BA}}} \texttt{h}$ if and only if for all `x` in $\mathcal{S}_{\mathcal{R}_A}$, $\texttt{g (x)} =_{\mathcal{S}_{\mathcal{R}_B}} \texttt{h (x)}$ (the equality between functions in ML is not computable in general).

Along the following sections, we will expose four different approaches proposed to produce the implementation of proofs in Homological Algebra. The notion of representation will allow us to illustrate the differences between them, as well as to explain the difficulties found in each one of them. First, we will introduce some ideas about the implementation of algebraic structures in Isabelle.

3.5 The symbolic approach

In order to explain in detail this symbolic approach, we will apply it to Lemma 2.2.14. First we enumerate the mathematical entities involved in its proof in Section 3.5.1. Then, we will explain the process of abstraction in Isabelle in Section 3.5.2, following the guidelines introduced in Section 3.4. Homomorphisms and endomorphisms will be considered simultaneously. Later, we will give an enumeration of which lemmas we have proved with these ideas, in Section 3.5.3, and finally we will detail the advantages and drawbacks of the method in Section 3.5.4.

3.5.1 The algebraic structures

In the statement of Lemma 2.2.14, the following mathematical entities appear explicitly[3]: the differential group (D, d_D), the differential subgroup $(\ker p, d_D)$, the abelian group endomorphisms h and 0_D, the abelian group homomorphism (which is also a differential group homomorphism) $d_D h + h d_D$, and the graded group homomorphisms $\mathrm{id}_{\mathrm{End}(D)}$ and $\mathrm{inc}_{\ker p}$.

Some other entities are not mentioned in the statement, but are used in the proof, and can be considered to appear implicitly; they are the rings of abelian group endomorphisms $\mathrm{End}(D)$, $\mathrm{End}(\ker p)$, the ring of differential group endomorphisms $\mathrm{End}((D, d_D))$, the abelian groups of abelian group homomorphisms $\mathrm{Hom}(D, \ker p)$, $\mathrm{Hom}(\ker p, D)$,

[3] As stated at the end of Chapter 2, ungraded structures will be used in the sequel.

the abelian groups of differential group homomorphisms $\mathrm{Hom}((D,d_D),(\ker p,d_D))$, $\mathrm{Hom}((\ker p,d_D),(D,d_D))$, and the ringoid formed by the abelian group and the differential group homomorphisms and endomorphisms of these structures.

In the symbolic approach, the model of the representation $\mathcal{M}_\mathcal{R}$, will be formed by the set of endomorphisms $\mathrm{End}(D)$. This means that the implementation of the algebraic structures D and $\ker p$ will not be given. Two reasons led us to choose this model:

- The group $\ker p$ is a subgroup of D, and this relation will allow us to represent some elements of $\mathrm{Hom}(D,\ker p)$, $\mathrm{Hom}(\ker p,D)$, $\mathrm{End}(\ker p)$ as elements of $\mathrm{End}(D)$; for instance, $0_{\mathrm{Hom}(D,\ker p)}$, which has a similar behaviour to $0_{\mathrm{End}(D)}$ in the sense that for any x in D, $0_{\mathrm{Hom}(D,\ker p)}(x)$ is equal to $0_{\mathrm{End}(D)}(x)$, could have the same representation in some situations.
- Some of the properties that have to be proved in Lemma 2.2.14 can be stated and proved just considering the properties of the ring $\mathrm{End}(D)$.

As far as our model considers just the set $\mathrm{End}(D)$, the rest of the elements of the framework will not have an explicit representation.

3.5.2 Representation of the algebraic structures and the homomorphisms

In this section, the process of representation in Isabelle of the mathematical setting exposed in Section 3.5.1 is described, following the definitions given in Section 3.4.

If we are only considering the properties of endomorphisms as elements of the ring $\mathrm{End}(D)$, the domain of the representation for the endomorphisms, $\mathcal{D}_R$, can be given in the form of elements of a variable type $'a$ in Isabelle, without any functional information.

Now, the carrier of the representation, $\mathcal{S}_R$, will be a set of elements over this type, $'a\ set$. At this stage, this type definition means that we will dispose of a predicate allowing us to know whether an object of $'a$ type is contained on this set or not. As we introduce operations and their properties over this set, it will be clear which elements belong to the carrier.

The algebraic structure that we use to encode endomorphisms is a ring of elements of $'a$ type. It will be implemented as explained in Section 1.3.2, i.e., through a record with five fields, one of them containing the carrier set, which corresponds to the carrier of the representation $\mathcal{S}_\mathcal{R}$, together with two constants *one* and *zero* and two binary operations *mult* and *sum*. With this implementation, the process of abstraction can be described as follows: any x in the carrier of the representation is interpreted through the abstraction $\Lambda_\mathcal{R}$ as a generic endomorphism of D. The process of abstraction is also such that *zero* is interpreted as the null endomorphism and *one* as the identity map from D to D. In addition to this, the abstraction process must respect the operations of the ring R, in such a way that for any x, y in R, $x \oplus y$ is interpreted as the addition in $\mathrm{End}(D)$ of the interpretation of x and y; in the case of the *mult* operation it is interpreted on $\mathrm{End}(D)$ as the composition of maps.

In order to fully describe the process of representation, the equality of the representation $=_{\mathcal{R}}$ must be defined. The equality of the representation corresponds to the equality in Isabelle/HOL. The equality in Isabelle is defined as an infix function with type

```
consts
  =      :: 'a => 'a => bool     (infixl 50)
```

It is declared to be reflexive, symmetric and transitive. Its definition is valid for functions and records, as we will see in the rest of approaches, but as far as in this approach we are dealing with objects of a generic type $'a$, we do not need yet any further information. The only important fact is that this equality can be augmented with the properties given for the ring algebraic structure (or for a concrete lemma). Therefore, if we have declared the ring R to satisfy associativity, we can also derive that given any x, y and z in R, $(x \otimes y) \otimes z = x \otimes (y \otimes z)$.

Whenever we state more premises in the lemmas, the equality of the representation will also include these information (for instance, if we define an element p in R to be idempotent, $p \otimes p$ and p will be equal through the equality of the representation).

The equality of the representation in this setting gives rise to what we will call from now on *equational reasoning*. We will understand by equational reasoning the kind of properties that can be proved from the premises of the algebraic structure where the objects are defined to be. *Reasoning by rewriting* could be another name for the same idea.

For instance, let us suppose that in our abstraction process the model of representation $\mathcal{M}_{R'}$ would be the set of endomorphisms over the integer numbers, $\mathrm{End}(\mathbb{Z})$, and that we consider again the carrier of the representation to be equal to the carrier of a generic ring R. The endomorphism $f(x) = x + 1 - 1$ in $\mathcal{M}_{R'}$ could be represented by a certain y in the carrier of R (without further information on it). Now, we have no way to prove, by using the representation proposed in this approach, that y, seen as an element of the ring R, is equal to 1_R, a property which is trivial in the model of the representation. The reason is that this property depends on the functional definition of f, and not on its properties as an element of the ring R.

In the following section we will describe the properties of Lemma 2.2.14 that can be proved in Isabelle by using this setting.

3.5.3 Lemmas proved in Isabelle

As we have already explained, this setting is quite coarse. As far as we have a representation only for the ring $\mathrm{End}(D)$ in the form of this generic ring R, we do not have a representation, first, of the concrete algebraic structures that the homomorphisms are connecting (in this case, the differential groups (D, d_D) and $(\ker p, d_D)$), and, second, about the way the concrete endomorphisms behave over a concrete element of their domain.

This implies that the lemmas that can be proved inside of this framework can depend only on the properties of the elements of $\mathrm{End}(D)$ interpreted as elements of a generic ring. In Lemma 2.2.14 a reduction is built starting from some given homomorphisms, and then it has to be proved that the five properties of reduction are satisfied. In particular, we are able to prove one of the properties needed in Lemma 2.2.14 inside of this framework. The fifth property is trivial, as far as it asserts one of the premises.

We can give a proof of the following equation, $(\mathrm{id}_{\mathrm{End}(D)} - p)h = 0_{\mathrm{Hom}(D,\ker p)}$, which corresponds to the third property of the reduction definition. The representation chosen for id_D is $\mathbf{1}$ of the ring R. The endomorphisms p, h and d_D are represented through generic elements of R, which we will name p, h and d. Finally, the homomorphism $0_{\mathrm{Hom}(D,\ker p)}$ is represented through the constant $\mathbf{0}$ in R. The representation of the properties of Lemma 2.2.14 remains as follows: $h \otimes d \otimes h = h$, $h \otimes h = \mathbf{0}$, and p is defined to be $d \otimes h \oplus h \otimes d$. Then, the proof of the third property of a reduction in Isabelle is as shown in Figure 3.1. In Figure 3.2 the mathematical proof of the same result is shown.

```
lemma (in ring) property-three-stepwise:
assumes D: d ∈ carrier R and H: h ∈ carrier R and H-H: h ⊗ h = 0
  and H-D-H: h ⊗ d ⊗ h = h and P: p = d ⊗ h ⊕ h ⊗ d
shows (1 ⊖ p) ⊗ h = 0
proof –
  from prems have (1 ⊖ p) ⊗ h = (1 ⊕ (⊖ p)) ⊗ h
    by (simp add: minus-def)
  also have ... = 1 ⊗ h ⊕ (⊖ p) ⊗ h
    by (rule l-distr) (simp-all add: prems)
  also from prems have ... = h ⊕ (⊖ p) ⊗ h
    by simp
  also from prems have ... = h ⊕ ⊖ (p ⊗ h)
    by (simp add: l-minus)
  also from prems have ... = h ⊕ ⊖ (h)
    proof –
      from prems have p ⊗ h = d ⊗ h ⊗ h ⊕ h ⊗ d ⊗ h
        by (simp add: l-distr)
      also from prems have ... = d ⊗ h ⊗ h ⊕ h
        by simp
      also from prems have ... = 0 ⊕ h
        by (simp add: m-assoc)
      also from prems have ... = h
        by simp
      finally show ?thesis by simp
    qed
  also from prems have ... = 0
    by (intro r-neg, intro H)
  finally show ?thesis by simp
qed
```

Figure 3.1: Proof in Isabelle

$$
\begin{aligned}
(\mathrm{id}_{\mathrm{End}(D_*)} - p)h &= h - ph \\
&= h - (dh + hd)h && \text{(unfolding the definition of } p\text{)} \\
&= h - hdh && \text{(using that } hh = 0_{\mathrm{End}(D_*)}\text{)} \\
&= h - h && \text{(introducing that } hdh = h\text{)} \\
&= 0_{\mathrm{Hom}(D_*, \ker(p))}
\end{aligned}
$$

Figure 3.2: Mathematical proof

It is worth noting that, even in this extremely simple example, the distance from the mechanized proof to the (highly) detailed mathematical proof is remarkable. The proof is written in a calculational reasoning style (see [Bauer and Wenzel, 2001]). The original expression in the statement is modified at each proof step (the symbol "..."

is an abbreviation in Isabelle meaning "the previous expression in the calculation"), in order to obtain the expression found in the goal of the lemma. When the goal is reached, the calculation is finished thanks to the use of the command "**finally**", by taking the original expression and the last one, and then, making use of transitivity of the equality, the property is proved. In the different steps of the Isabelle proof, only what we have defined as equational reasoning has been applied, (and, in fact, in this approach, this is the *unique* available tool).

The lemma can be proved in Isabelle in a much more automated way, simply by applying one tactic:

lemma (**in** *ring*) *property-three*:
  assumes $d \in$ *carrier* R **and** $h \in$ *carrier* R **and** $h \otimes h = \mathbf{0}$ **and** $h \otimes d \otimes h = h$ **and** $p = d \otimes h \oplus h \otimes d$
  shows $(\mathbf{1} \ominus p) \otimes h = \mathbf{0}$
proof –
  from *prems* **show** *?thesis* **by** *algebra*
qed

The proof is obtained in Isabelle by introducing the premises and then applying the *algebra* tactic, defined by C. Ballarin. This tactic is special for algebraic structures, and tries to apply simplifications derived from the inherent properties of the structure (in this case, a ring), as well as from the premises, in order to unify both sides of an equation. In this case, the premises and also the previous knowledge in the Isabelle library for rings are enough to automatically unify both sides and therefore to prove the lemma.

3.5.4 Discussion

Our first approach consists in representing the objects in the statement of the lemmas (the homomorphisms) as generic elements of an algebraic structure where equational reasoning can be carried out. The idea is to represent the elements of the ring of endomorphisms as elements of a generic ring. Then, calculations in this ring are identified with proof steps in the reasoning domain of the endomorphisms. We call this a *symbolic approach* since endomorphisms are represented by symbols (as elements of a generic ring) without reference to their functional nature.

The proof of Lemma 2.2.14 illustrates most of the problems that have to be solved in order to have a framework for the collection of lemmas in Section 2.2.2: the implementation of complex algebraic structures and the implementation of homomorphisms. In addition, one must work with homomorphisms at two different abstraction layers simultaneously: equationally, like elements of an algebraic structure (in order to increase the amount of previous knowledge for proofs, by means of the properties of that algebraic structure), and also like functions over a concrete domain and codomain (in order to be able to reason with elements of the domain, and thus, be able to complete proofs).

In our process of representation, we have defined the abstraction Λ identifying elements of a generic ring R with the endomorphisms of (D, d_D). Therefore, there is no representation of the algebraic structures (D, d_D) and $(\ker p, d_D)$. It could be tried to represent some objects of these structures as elements of the ring R. Actually, this is what has been done in the case of the third property, whose Isabelle proof was exhibited in Section 3.5.3; an element of $\operatorname{Hom}(D, \ker p)$ is represented as an element of the ring R. More concretely, we have represented the element $0_{\operatorname{Hom}(D,\ker p)}$ through the element $\mathbf{0}$ of the ring R. Two different representation contexts have been used at the same time. The first one, as explained, identifying the elements of $\operatorname{End}(D)$ with the elements of the carrier of the ring R. The second one, identifying also elements of $\operatorname{Hom}(D, \ker p)$ with elements of the ring R. In this case, the combination of both representations is coherent.

On the other hand, if we try to represent other properties in Lemma 2.2.14 in the same way, we will observe that identifying several processes of representation with a same carrier can produce unexpected results. For instance, if we consider the first property, stating that $(\mathrm{id}_{\operatorname{End}(D)} - p) \operatorname{inc}_{\ker p} = \mathrm{id}_{\operatorname{End}(\ker p)}$, the elements $\mathrm{id}_{\operatorname{End}(D)}$ and p, which belong to the set $\operatorname{End}(D)$, can be represented as elements in the carrier of R; if we try to consider as our model of representation the set $\operatorname{Hom}(D, \ker p)$, with carrier again the carrier of R, a natural representation for $\operatorname{inc}_{\ker p}$ would be to consider it as the element $\mathbf{1}$ of R; finally, if we try to consider the same carrier of representation for the ring $\operatorname{End}(\ker p)$, then $\mathrm{id}_{\operatorname{End}(\ker p)}$ would be represented also as $\mathbf{1}$ in R. We have used the same carrier of the representation for objects of three different models of representation. This leads us to produce a wrong representation of the equation we want to prove. The representation of the property would look like as $(\mathbf{1} - p) \otimes \mathbf{1} = \mathbf{1}$, which is unprovable in our context (i.e., with the stated premises).

This symbolic approach has some advantages. First of all, it is quite simple and direct, which has as a consequence that proofs are readable and quite close to the mathematical proofs obtained, which we consider must be one of the goals when using a proof assistant. As will be seen in Section 3.6, where more elaborate approaches are discussed, it is also possible that the size of the proofs turns them into something unfeasible and of little help for the working mathematician. Moreover, Isabelle has among its standard libraries enough theories to produce proofs in the context of the symbolic approach in a completely automatized way; these tactics are specially efficient when only equational reasoning is required, arriving in some cases to fully automatize the proof, as in the example shown with the *algebra* tactic.

The most appealing idea found in this method was to introduce in the framework the algebraic structures formed by the homomorphisms (i.e., the ring $\operatorname{End}(D)$). In this approach the idea was introduced, and it will be further developed in the morphism based approach in Section 3.7 and also in the interpreting locales approach in Section 3.8.

There are also drawbacks on this method. Firstly, we cannot prove the remaining properties (the first, second and fourth ones in the reduction definition applied to the reduction built in Lemma 2.2.14) needed to complete the implementation of the proof.

They can be neither proved nor even expressed inside of this framework, since information about the domain of the homomorphisms or about the concrete definition of the homomorphisms in such domains is required. For instance, it is not possible to derive with the tools of this framework that $p\big|_{(\ker p)} = 0_{\mathrm{End}(\ker p)}$. This is a consequence of the abstraction process that we have chosen for this framework. The representation is too generic. The conceptual distance between Isabelle code and the mathematical content is too large.

To sum up, the representation is interesting, illustrating some important points, but the domain of the representation was conceptually too far from the mathematical model, and it was not possible to implement every proof or property. The previous discussion shows that the computational content of homomorphisms (that is to say, their interpretation as functions) is lost, which prevents code extraction. These drawbacks are overcome in the following approaches.

3.6 The set theoretic approach

As we have pointed out, the framework presented in the previous section has mainly two weak points. First, the representation chosen was correct, but some limitations were found when dealing with homomorphisms and elements of algebraic structures. Second, and based on the previous weakness, some properties and proofs could not be expressed and proved with the given tools.

In this section we detail a framework based on the representation of mathematical entities given in the Isabelle libraries. We also face the implementation of the complete proof of both Lemmas 2.2.11 and 2.2.14. Despite the similarities between their statements, a complete proof of the first one will be shown with the representation given in this set theoretic approach, whereas some limitations appeared when trying to implement the proof of Lemma 2.2.14 that will make us propose a new setting. We point out the differences between both lemmas in Section 3.6.5, paying special attention to the possible solutions that will lead us to the morphism based approach and the interpreting locales approach.

3.6.1 The algebraic structures

We first describe which of the mathematical structures appearing (explicitly or implicitly) in Lemmas 2.2.11 and 2.2.14 are going to be considered in our model of representation. In the previous approach, only one model of representation (the set of endomorphisms) was considered, and we highlighted the consequences of introducing more than one model with the same carrier of the representation. Here, various processes of representation are going to be considered with different models of representation and their respective carriers sets; all of them will have to interact in an appropriate way.

In the statement of Lemma 2.2.14 the following structures appear explicitly (as related in Section 3.5.1): the differential group (D, d_D), the differential subgroup $(\ker p, d_D)$, the abelian group endomorphisms h, 0_D, the abelian group homomorphism $d_D h + h d_D$, and the abelian group homomorphisms $\mathrm{id}_{\mathrm{End}(D)}$ and $\mathrm{inc}_{\ker p}$. Some properties of the following algebraic structures are also used: the rings $\mathrm{End}(D)$, $\mathrm{End}(\ker p)$, the ring $\mathrm{End}((D, d_D))$, the abelian groups $\mathrm{Hom}(D, \ker p)$, $\mathrm{Hom}(\ker p, D)$, the abelian groups $\mathrm{Hom}((D, d_D), (\ker p, d_D))$, $\mathrm{Hom}((\ker p, d_D), (D, d_D))$, and the ringoid formed taking the abelian group endomorphisms and homomorphisms of these structures.

In the statement of Lemma 2.2.11, the differential group (D, d_D) with its differential subgroup $(\ker gf, d_D)$, as well as (C, d_C) with its differential subgroup $(\operatorname{im} gf, d_C)$ appear. Moreover, the reduction $(f, g, h)\colon (D, d_D) \Rightarrow (C, d_C)$ is also explicit in the statement, and finally, the abelian groups $\mathrm{Hom}(D, \ker gf)$, $\mathrm{Hom}(\ker gf, D)$ appear in the proof.

In the implementation of the proofs of Lemma 2.2.14 in the set theoretic approach a representation of the following mathematical entities will be given:

- The differential groups (D, d_D) and $(\ker p, d_D)$ (whereas in the symbolic approach no explicit representation of them was given).
- The differential group endomorphisms $d_D\colon (D, d_D) \to (D, d_D)$ and $\mathrm{id}_{\mathrm{End}(D)}\colon (D, d_D) \to (D, d_D)$, the abelian group homomorphism $h\colon D \to D$, and combinations of them (for instance, $p = d_D h + h d_D$) (as in the symbolic approach, although their representation is different).
- The *sets* of abelian group homomorphisms $\mathrm{Hom}(D, \ker p)$ and $\mathrm{Hom}(\ker p, D)$, the set of endomorphisms $\mathrm{End}(D)$, $\mathrm{End}((D, d_D))$, $\mathrm{End}(\ker p)$ and $\mathrm{End}(\ker p, d_D)$, and the sets of differential group homomorphisms $\mathrm{Hom}((D, d_D), (\ker p, d_D))$ and $\mathrm{Hom}((\ker p, d_D), (D, d_D))$ (in the previous approach, only the ring $\mathrm{End}((D, d_D))$ was represented by means of a generic *ring*).

A similar situation is produced when we define a representation for Lemma 2.2.11, where the chosen items are:

- The differential groups (D, d_D), (C, d_C) and $(\operatorname{im} gf, d_C)$.
- The differential group homomorphisms $f\colon (D, d_D) \to (C, d_C)$ and $g\colon (C, d_C) \to (D, d_D)$.
- The *sets* of differential group homomorphisms $\mathrm{Hom}((D, d_D), (C, d_C))$, $\mathrm{Hom}((C, d_C), (D, d_D))$, $\mathrm{Hom}((\operatorname{im} gf, d_C), (C, d_C))$, and $\mathrm{Hom}((C, d_C), (\operatorname{im} gf, d_C))$.

As it can be observed, the main differences with respect to the symbolic approach is that the domain and codomain of homomorphisms are included, and also that homomorphisms will be considered as elements of *sets* instead of as elements of some algebraic structure. These design decisions are based on the encoding proposed in the Isabelle libraries.

3.6.2 Representation of the algebraic structures

We consider first the mathematical entities appearing in the representation of Lemma 2.2.14. In this section we describe the representation of algebraic structures, mainly differential groups, and in Section 3.6.3 we will face the problem of the representation of homomorphisms.

The representation chosen for differential groups is connected to the implementation proposed in Section 1.3.2 through records, where a carrier field represents the set of elements of the algebraic structure, and the rest of fields represent the constants and operations of the algebraic structure. We describe the representation process with an example, identifying the elements exposed in Section 3.4 appearing in the representation process in the case of a generic differential group. Let us consider a differential group (D, d_D). It will be the model of representation $\mathcal{D}_R$ of our example. It can be thought of as a set of elements with some common operations. We use as domain of the representation $\mathcal{S}_R$, the sets over a variable type in Isabelle; it can be observed that this data type corresponds to the data type of the only field of the defined record *partial-object* (see Section 1.3.2), which is the basic record type defined for algebraic structures, from which the rest of algebraic structures are extended (or inherited). The operations and constants of the differential group are represented with objects of a functional type with the corresponding arities. In addition to this, some assumptions ensuring the right properties of the differential group structure are provided (see Section 1.3.2); these assumptions must be satisfied by a record in order to satisfy the *differential-group* predicate. Then, the carrier of the representation $\mathcal{S}_R$ will be an Isabelle object with type $'a\ set$ such that together with the given operations satisfy the properties of the algebraic structure. With this carrier, the abstraction can be defined as a function from the Isabelle *carrier* into the carrier of our representation model such that **1** in Isabelle is assigned to the unity object of $\mathcal{M}_R$. Finally, the equality of the representation now corresponds to the Isabelle equality which has been augmented with the properties derived from the definition given in Isabelle for the differential group algebraic structure.

Some other algebraic structures appear in the statement of the lemmas, as the kernel and the image sets of an application. The Isabelle definition of the image set can be found in the Isabelle library, more concretely in the file theory *Set* and is as follows:

constdefs
  $image :: ('a => 'b\ ) => 'a\ \ set => 'b\ set$ **(infixr** ‘ 90)
  f ‘ $A\ \ == \{y.\ EX\ x{:}A.\ y = f\ (x)\}$

The image set is formed by the elements with a preimage. In a similar way we had to define the kernel set:

constdefs
  $Ker :: [('a,'c)monoid\text{-}scheme,\ ('b,\ 'd)monoid\text{-}scheme,\ 'a => 'b\ ] => 'a\ set$
  $Ker\ G\ G'\ f\ == \{x.\ (x \in carrier\ G) \wedge\ f\ x = one\ G'\}$

The kernel is defined for objects with type, at least, *monoid*, which ensures that we have a *one* field, making the definition meaningful.

3.6.3 Homomorphisms between algebraic structures

In this setting, we will paid an special attention to the representation of homomorphisms. First we describe the codification of functions between sets in Isabelle, based on Example 3.4.6. Then, we will comment on the Isabelle technicalities of the implementation of homomorphisms.

As far as our representation of algebraic structures is formed by sets with some operations, the representation of homomorphisms between algebraic structures can be seen as the problem of representing functions between sets. Let us choose two sets A and B. The model of representation is the same chosen in Example 3.4.6, i.e., the set of functions B^A. The domain of our representation is defined to be the type of total functions between the types of the representation of sets A and B[4]; here we consider two generic types $'a$ and $'b$ (in ML they were named α and β). Then, the domain of the representation will be the data type $'a =>' b$. The carrier of the representation contains the terms of the data type $'a =>' b$ which satisfy the Isabelle axiom $h \in A -> B$. The meaning of the arrow in the previous expression will be later detailed in this section, for now it is enough to say that it defines the set of functions such that map every element of the Isabelle set A to an element of the Isabelle set B. The abstraction function maps an Isabelle function f between two sets A and B to a similar function of the model of representation. Here, a problem can be pointed out in the abstraction function. In general, it is not injective. Sets in Isabelle, as already pointed out, are partial objects over a type, i.e., if we consider a variable type $'a$ for representing a set A, its representation will be a term of type $'a\,set$. Now, taking into account that equality between functions in Isabelle is defined to be *extensional*, two functions f and g will be equal iff

$$f = g \iff \forall x. fx = gx$$

In Isabelle, and thanks to the extensionality principle, we can say that two functions are equal if they yield the same values. For instance, the following lemma can be proved in Isabelle in a simple proof step:

lemma *equal−functions*: **assumes** $f = (\lambda x{::}int.\ x + 1 - 1)$ **and** $g = (\lambda x.\ x)$ **shows** $f = g$
by (*simp add*: *prems*)

In the lemma, two elements of End($\mathbb{Z}$) are defined, $\lambda x.x + 1 - 1$ and $\lambda x.x$, and are proved to be equal. In the symbolic approach, the same example was introduced. Nevertheless, due to the process of representation chosen there, this equality between

[4]In HOL, the symbol => represents only total functions.

functions was not provable. Therefore, the representation proposed in this approach has greater expressiveness.

On the contrary, some problems also arise from working with functional objects. Sets are represented as partial objects over a given type; then, if we define two functions between sets, they will be equal if they are equal when applied to every element of their source type, and not of the carrier of their source set. Consequently, the following statements are not provable in Isabelle[5]:

assumes $f \in \{0::int,\ 1\} \to \{0,\ 1\}$ **and** $f\,(0) = 0$ **and** $f\,(1) = 1$ **shows** $f = id$

In this statement, a function is defined from a set representing $\mathbb{Z}_2$ into this same set, mapping each element to itself. Then, we try to compare it to the identity function in Isabelle. Due to the extensionality, these two functions will be equal if they are equal for every value of the data type *int*. This cannot be proved in this case.

The same statement can be proposed in a generic setting, being A an B two sets and f a function between them, with the same result. The equality is not provable:

assumes $f \in A \to B$ **and** $g \in A \to B$ **and** $\forall\ x \in A.\ f\,x = g\,x$ **shows** $f = g$

As a consequence of these facts, injectivity of the abstraction function for functions between sets has been lost. When we deal with functions between sets, a function in the model of the representation can have several preimages in the carrier of the representation. Some solutions for this problem will be commented on in Section 3.6.5, and one of these solutions will be applied in our next setting, the morphism based approach, in Section 3.6.

After describing the process of representation for functions between sets in this setting, we now describe in detail how homomorphisms are implemented in Isabelle. A type declaration and a collection of axioms must be provided to have an implementation of homomorphisms. The type will be defined depending on two parameters, namely the algebraic structures representing the domain (or source) and the codomain (or target) of such a homomorphism. Let us suppose that G and H are two semigroups, defined through the record types $('a,'c)$ *semigroup-scheme* and $('b,'d)$ *semigroup-scheme*. The type of the Isabelle set *hom G H* (which is the carrier of the representation of the set $\mathrm{Hom}(G, H)$ defined in Section 1.1.2), will be defined to be the set of functions connecting the types of G and H:

consts
  hom :: $[('a,\ 'c)$ *semigroup-scheme*, $('b,\ 'd)$ *semigroup-scheme*$]\ \ => ('a => 'b)$*set*

The type described defines the set of functions such that given an element of the type

[5]The following statement has not been named with the Isabelle keyword *lemma* since it is not provable.

of the carrier set of G return an element of the type of the carrier set of H. A relevant fact that can be pointed out is the use of schematic record types. The type definition above is valid for semigroup record types, but also for any record subtype obtained from this one; for instance, monoid record types and differential group record types, thanks to extensible records.

Once that the set of homomorphisms has been given a type, its definition (in terms of HOL logic) must be given. Information has to be provided in order to ensure that homomorphisms are well-defined for the carrier sets of the algebraic structures. The axioms allowing us to distinguish which total functions satisfy the homomorphisms' properties are

```
defs
  {h. h ∈ carrier G −> carrier H &
    (∀ x ∈ carrier G. ∀ y ∈ carrier G. h (mult G x y) = mult H (h x) (h y))}
```

The second part of the definition establish the compatibility of the homomorphism with the multiplicative operation in the semigroup record. The partiality is contained in the first axiom stating that $h \in \mathit{carrier}\, G \mathrel{-\!\!>} \mathit{carrier}\, H$.

The arrow −> represents in Isabelle a simplified version of the Π-type for dependent sets. Homomorphisms, interpreted as functions between sets over types, can be seen as a set of functions such that when we have an element in the source set, its image will be an element of the image set:

```
syntax
  funcset :: ['a set, 'b set] => ('a => 'b) set      (infixr −> 60)
```

In order to get pretty printing facilities, first a symbol −> is defined to be equivalent to the operator *funcset* for two sets. Given two sets A and B, the following lemma proved in Isabelle shows the equivalence between the *funcset* function and the set of functions which image will be an element of a set B whenever their source is an element of A.

```
lemma funcset-definition: shows funcset A B = {f. ∀ x. x ∈ A −−> f x ∈ B }
  by (auto simp add: funcset-mem, unfold Pi-def, simp)
```

It can be observed that this definition fits into the prerequisites of partiality between sets that we asked for the carrier sets of algebraic structures. Nevertheless, with this definition, the process of abstraction for homomorphisms lacks of injectivity, allowing several functions from the carrier of the representation to represent the same mathematical entity. Consequently, the difficulties of defining an algebraic structure with this set of functions can be observed.

Mainly two operations between homomorphisms and endomorphisms appear in the statements of lemmas in Section 2.2.2. One of them is represented as the product, and can be identified with the composition of functions. In Isabelle composition of functions is defined as:

constdefs
  $comp :: ['b => 'c, 'a => 'b, 'a] => 'c$ (**infixl** $\circ$ *55*)
  $f \circ g == \lambda x.f\,(g(x))$

Some properties have been already proved about this definition that will result helpful for improving the mechanization and decrease the size of the Isabelle proofs. However, and quoting what is said in [Nipkow et al., 2002] referred to function theory in Isabelle, "unlike with set theory, however, we cannot simply state lemmas and expect them to be proved using `blast`", where `blast` is the name of one of the tactics in Isabelle, which is based on tableau methods. The other relevant operation with endomorphisms in Isabelle is addition. For instance, one of the properties of reduction in Lemma 2.2.14 states that $\mathrm{inc}_{\ker p}(\mathrm{id}_D - p) + d_D h + h d_D = \mathrm{id}_D$. Addition of endomorphisms will depend on the binary operation of the underlying algebraic structure. In this case, for instance, the addition in D. Contrarily to the composition, this operation is not predefined in Isabelle, and no facts are available in Isabelle libraries. The way to define the endomorphism formed by the addition of two endomorphisms is by means of a lambda definition. The following Isabelle function:

$(\lambda x.\ add\ D\ (id\ x)\ (h\ x))$

denotes in Isabelle, with the existing facilities, the endomorphism $\mathrm{id}_D + h$ in the set of endomorphisms $\mathrm{End}((D, d_D))$[6].

From this operation it has to be also defined the difference between endomorphisms. As we will see in Section 3.6.4, when we try to implement proofs with these tools, expressions become untidy and goals split up very quickly.

3.6.4 Lemmas proved

With the ideas and the chosen representations explained in Sections 3.6.1, 3.6.2 and 3.6.3, we first faced the implementation of the proof of Lemma 2.2.11, in which we succeeded, and then we tried, as in the symbolic approach, the complete implementation of the proof of Lemma 2.2.14, where we did not succeeded. In this section, we first report on the proof of Lemma 2.2.11 and then on the problems found when implementing Lemma 2.2.14.

In the statement of Lemma 2.2.11, an isomorphism between the differential groups (C, d_C) and $(\mathrm{im}\, gf, d_D)$, being (f, g, h) a reduction from (D, d_D) to (C, d_C) is defined

[6]Note that the additive operation of the ring $\mathrm{End}((D, d_D))$ is defined from the additive operation of the differential group (D, d_D).

by means of $f\colon (\operatorname{im} gf, d_D) \to (C, d_C)$ and $g\colon (C, d_C) \to (\operatorname{im} gf, d_D)$. The proof, as presented in Lemma 2.2.11, is divided into two parts. First, it must be checked that $(\operatorname{im} gf, d_C)$ is a differential group. Then, it must be proved that f and g define an isomorphism between the differential groups (C, d_C) and $(\operatorname{im} gf, d_D)$. From the definition of reduction applied to (f, g, h), f is defined as a differential group homomorphism $f\colon (D, d_D) \to (C, d_C)$. In order to define the isomorphism between differential groups, we must also consider it as $f\colon (\operatorname{im} gf, d_D) \to (C, d_C)$.

The proof in Isabelle can be done following the same pattern. We first prove that $(\operatorname{im} gf, d_D)$ is a differential subgroup of (D, d_D), and then we prove that the proposed maps define an isomorphism. Actually, in the first part, what we prove is a more general result, stating that whenever we have a homomorphism between two differential groups, for instance $i\colon (C, d_C) \to (D, d_D)$, the image set of i together with the inherited operations of (D, d_D) form a differential group, which corresponds to Proposition 2.2.10.

We take advantage for this proof from the inheritance between algebraic structures. We first prove the property for groups and homomorphisms between groups, and then, adding the required axioms, we extend it also to abelian groups and finally to the case of differential groups. The statement of the property in Isabelle for groups is as follows:

```
lemma hom-image-is-subgroup: ⟦ group C; group D; f ∈ hom C D ⟧
  ⟹ subgroup (f‘ (carrier C)) D
```

where it is stated that the image set through f of the carrier set of G will be a subgroup of D.

The definition of subgroup in Isabelle is derived from the one of submagma. A submagma is a subset of a given structure which also is closed under the multiplicative operation. Then, a subgroup is defined to be a submagma which, in addition to this, contains the unit of the structure from which it is a subgroup, and is closed for the inverse function. There is another characterization of subgroups, which is the one we use in order to prove the previous statement. It requires that the subgroup must be a non-empty subset, which is closed under multiplication and inverse operations.

These are the premises that we prove in Isabelle. The Isabelle degree of mechanization allows us to combine pieces of code where subgoals can be discarded with a high degree of automation, for instance, that the image set of f is a non-empty set:

```
assume D: group D
assume C: group C
assume f: f ∈ hom C D
 show Not-empty: f ‘ carrier C ≠ {}
  proof -
    from C   have one C ∈ carrier C by (simp add: group-def [of C] monoid-def)
    from this have f (one C) ∈ f ‘ carrier C by (simp add: imageI)
    then show ?thesis by force
  qed
```

where the subgoal is proved in three proof steps (that involve several tactics). Here it has been used that the image through f of the unit in C is an element of the image set. Thus, the set is non-empty. On the contrary, there are another pieces of code, for instance calculations, where reasoning has to be done stepwise and mainly guided:

```
show mult D x y ∈ f ‘ carrier C
 proof –
   from X obtain x′ where x-image: x = f x′ ∧ x′ ∈ carrier C by blast
   from Y obtain y′ where y-image: y = f y′ ∧ y′ ∈ carrier C by blast
   from C and x-image and y-image
   have mult-in-C: mult C x′ y′ ∈ carrier C
     by (simp add: group-def [of C] magma.m-closed [of C])
   have (mult D x  y) = f (mult C x′ y′)
   proof –
     from x-image and F have x-in-D: x ∈ carrier D
       by (simp add: hom-closed [of f])
     from y-image and F have y-in-D: y ∈ carrier D
       by (simp add: hom-closed [of f])
     from D and C and x-image and y-image and x-in-D and y-in-D and F
     have (mult D x  y) = mult D (f x′) (f y′)
       by simp
     also have ... = f (mult C x′ y′)
       by (simp only: F x-image y-image hom-mult [of f C D x′ y′])
     finally show ?thesis
       by simp
   qed
   then show ?thesis by (simp add: mult-in-C imageI)
  qed
```

Here, automatized tactics cannot succeed to find directly the matching rules stating that for any x and y in the image set of f (which are, thus, elements of the carrier of D), $mult\,D\,x\,y \in carrier\,D$. It must be proved that x and y are of the form fx' and fy' and then the proof is carried out in the group C with x' and y', being finally transferred to D with x and y. The prover must be fully guided. Nevertheless, with a careful development of the proof steps, and using in a restricted way the automatized tactics, the proof can be completed.

Once we have proved that the image set of a homomorphism is a subgroup, the proof of Lemma 2.2.11 can be faced. It must be proved that an isomorphism can be defined between the given group C and the subgroup of D, $\operatorname{im} gf$ through $f\colon (\operatorname{im} gf, d_D) \to (C, d_C)$ and $g\colon (C, d_C) \to (\operatorname{im} gf, d_D)$. Taking advantage of the Isabelle facilities for later introducing inheritance both in the axiomatic part and also in the record types, we will be able to re-use this lemma when proving its version for abelian groups and differential groups.

In the Isabelle2004 libraries cannot be found a definition of isomorphic groups and therefore we had to introduce it. From the different equivalent definitions, and bearing in mind the statement of Lemma 2.2.11, we decided to use the following one:

locale *isomorph-groups* = *group* G + *group* G' +
  assumes *isomorphGG'*:*isomorph-semigroups* G G'

constdefs
  isomorph-semigroups :: [('a,'c)*semigroup-scheme*, ('b,'d)*semigroup-scheme*] => *bool*
  isomorph-semigroups S S' == ∃ f ∈ (*hom* S S'). ∃ g ∈ (*hom* S' S).
    (*restrict* ($f \circ g$) (*carrier* S') = *restrict id* (*carrier* S'))
    ∧ (*restrict* ($g \circ f$) (*carrier* S) = *restrict id* (*carrier* S))

The definition is equivalent to the one of isomorphic semigroups, as already happened with the definition of homomorphism, which also were equivalent (just the algebraic structures involved in the definition must be modified).

Two semigroups S and S' will be isomorphic whenever two homomorphisms $f\colon S \to S'$ and $g\colon S' \to S$ can be found, such that their compositions in both directions are equal to the corresponding identities, i.e., id_S and $\mathrm{id}_{S'}$.

In the Isabelle definition of isomorphic semigroups it can be observed some of the consequences derived from the implementation of homomorphisms through total functions over types. Due to the definition of homomorphisms in Isabelle, we know that they satisfy some properties inside of their domain set; on the contrary, we do not have any information about their behavior with respect to the rest of the elements of their source type. This situation forces us to use the Isabelle predefined function *restrict*:

constdefs
  restrict :: ['a => 'b, 'a *set*] => ('a => 'b)
  restrict f A == (λx. *if* $x \in A$ *then* f x *else arbitrary*)

It allows us to study the behavior of the function just for the elements of the source set, and not in the whole range of elements of the type of the source set. This problem, as well as some possible solutions, will be studied in detail in Section 3.6.5.

The reserved word *arbitrary* stands for a polymorphic constant valid for any type which is fully unspecified, and it represents undefined terms (for instance, the result of applying a function to an element out of the function domain).

Once that all these technical considerations have been made, the statement of Lemma 2.2.14 in its version for groups can be given:

lemma *isomorphism-version-groups*:
  ⟦ *group* C; *group* D; f ∈ *hom* D C; g ∈ *hom* C D ;
  ⋀ x. x ∈ *carrier* C ⟹ ($f \circ g$) x = *id* x ;
  ($g \circ f$) ∘ ($g \circ f$) = ($g \circ f$)⟧

$\Longrightarrow$ (*isomorph-groups C* (*D* (| *carrier* := *image* ($g \circ f$) (*carrier D*) |)))

All the premises have been directly extracted from the definition of reduction applied to the groups C and D and the tuple (f, g) (here the homotopy operator can be omitted). The goal of this lemma consists in proving the isomorphisms between C and the subgroup of D with carrier set $\operatorname{im} gf$.

Now the proof can be implemented in Isabelle in a nearly mathematical style. First, both algebraic structures, C and $\operatorname{im} gf$ have to be shown to be groups. Then, two homomorphisms have to be defined satisfying the isomorphism definition. Then, they have to be proved to be well defined, and finally, their compositions will have to produce the identity in both directions.

The algebraic structure C is known to be a group directly from the premises. The set $\operatorname{im} gf$ with the operations of D is a group from the previous Isabelle lemma *hom-image-is-subgroup*, which stated that for any homomorphism f, the subset $\operatorname{im} f$ is a subgroup, applied to the homomorphism (gf) and the group D. The homomorphisms can be explicitly built thanks to the Isar capacity for fixing schematic variables inside of a proof by using the ***let*** command:

let *?g* = *restrict g* (*carrier C*)
let *?f* = *restrict f* (*image* ($g \circ f$) (*carrier D*))

Later, we will be able to instantiate the existentially quantified variables that appear when unfolding the definition of isomorphic groups with the schematic variables *?f* and *?g* that we have defined.

Again, in the definition of *?f* and *?g* can be observed the necessity of substituting the homomorphisms f and gf by their restrictions to the carrier sets of the isomorphic structures. This ensures the correctness of the compositions of *?f* and *?g*; maybe the use of the restrict operator is redundant in the case of *?g*, which is restricted to its own source set, since the restrictions already appear in the isomorphism definition. In the case of *?f* the use of the restriction is needed, since gf has as source the group D while *?f* is the result of the restriction of gf to the set $\operatorname{im}(gf)$.

The rest of the proof is quite straightforward. Due to the partiality matters introduced by the definition of the homomorphisms as partial functions over types, goals usually split into different cases depending if we are inside of the source sets or not. Therefore, complementary goals appear, that sometimes Isabelle automatically discards (or proves), but sometimes have to be carefully handle. The whole proof is around 330 lines of Isabelle code.

Once we have proved the lemma for a pair of groups D and C, we change the statement to the case where D and C are abelian groups. The type definition and axioms for the homomorphisms between groups and for homomorphisms between abelian groups are equal, and then in the statement for abelian groups just the premises about C and

D are modified[7]:

lemma *isomorphism-version-abelian-groups*:
$\llbracket$ *abelian-group* C; *abelian-group* D; $f \in$ *hom* D C; $g \in$ *hom* C D ;
$\bigwedge x.\ x \in$ *carrier* $C \Longrightarrow (f \circ g)\ x =$ *id* x; $(g \circ f) \circ (g \circ f) = (g \circ f)\rrbracket$
$\Longrightarrow$ (*isomorph-ab-groups* C (D (| *carrier* := *image* $(g \circ f)$ (*carrier* D) |)))

Introducing the previous lemma where this property was proved for C and D groups, the only condition that remains to be proved is the one stating that the image set im gf is an abelian subgroup of the abelian group D, which in a more general way can be stated as:

lemma *ab-subgr-is-ab-group*: $\llbracket$ *abelian-group* A; *subgroup* H A $\rrbracket$
$\Longrightarrow$ *abelian-group* (A(| *carrier* := H|))

None of these lemmas requires a significant amount of code to be proved, due to the similarities between abelian groups and groups.

Once we have a version for abelian groups, the following step will consist in substituting the abelian groups by differential groups:

lemma *reduction-implies-isomorph-diff-groups*:
assumes *diff-group* CC **and** *diff-group* DD **and** $f \in$ *hom-cc* DD CC
and $g \in$ *hom-cc* CC DD **and** $\bigwedge x.\ x \in$ *carrier* CC ==> (f *o* g) x = *id* x
and (g *o* f) *o* (g *o* f) = (g *o* f)
shows (*isomorph-diff-groups* CC (DD(| *carrier* := *image* (g *o* f) (*carrier* DD) |)))

The proof for this version is based, following the general pattern, on the one for abelian groups. Nevertheless, some differences between both versions required some additional lemmas. First, a new definition of homomorphisms between differential groups had to be provided; as far as homomorphisms between differential groups require an additional property than homomorphisms between semigroups, this must be considered in the definition. Consequently, the definition of isomorphic differential groups was also changed, being defined in terms of homomorphisms between differential groups. Two auxiliary lemmas were also extracted from the main lemma to simplify the proof. The first one states that the image set of a differential group homomorphism is a differential subgroup, with the inherited operations of the source differential group:

lemma *hom-cc-image-is-diff-group*:
assumes *diff-group-CC*: *diff-group* CC **and** *f-hom-cc*: $f \in$ *hom-cc* CC CC
shows *diff-group* (CC (| *carrier* := *image* f (*carrier* CC) |))

[7] As pointed out in Section 1.3.2, when dealing with abelian structures, the operations *mult*, *one* and *m-inv* become *add*, *zero* and *a-inv*.

In the proof of this lemma, the previous lemmas *hom-image-is-subgroup* and *ab-subgr-is-ab-group* were introduced (by instantiating them with the corresponding values).

An important feature of theorem proving, detection of erroneous statements, showed its usefulness in our first attempt to prove this lemma. We intended to prove a more general result, similar to lemma *ab-subgroup-is-ab-group*, but applied to differential groups. The statement would be as follows:

$$[\![\ \textit{diff-group}\ A;\ \textit{subgroup}\ H\ A\ ]\!] \Longrightarrow \textit{diff-group}\ (A(\!|\ \textit{carrier} := H|\!))$$

Isabelle can not infer by itself that the result is erroneous; when one tries to give a mechanized proof for the previous statement, the goal is split up in (simpler) subgoals. This process, sooner or later, produces a subgoal of the main goal which can be clearly identified as a false statement (which maybe had not been observed in the original statement). This process, obviously, must be ruled by a human agent, and is included in the interactive part of the proving process. The user is responsible for identifying a false statement. Nevertheless, a lot of the Isabelle tactics do not preserve provability, just a small subset known as "safe" does, which means that obtaining a non-provable subgoal not always mean that we have a non-provable (initial) goal.

The second previous lemma proves that the composition of homomorphisms between differential groups is again a homomorphism between differential groups.

The whole proof, including some previous facts, definitions and lemmas, can be found in the theory files *"lemma1_previous.thy"*, *"lemma1_isom_groups.thy"*, *"lemma1_isom_ab_groups.thy"* and *"lemma1_isom_cc.thy"*; it is about 1700 lines long.

Now we can compare the proved lemma in Isabelle, *reduction-implies-isomorph-diff-groups*, and the mathematical version given in Lemma 2.2.11. Two differences can be observed. First, as we have already explained, the substitution of graded structures (chain complexes) by ungraded ones (differential groups). The second difference is that, instead of providing Isabelle with a definition of reduction, we preferred to just include between the premises the needed properties from the reduction definition. The main motivation for this was to obtain a more readable statement of the lemma in Isabelle. Apart from these two points, the version proved in Isabelle can be directly coupled with the mathematical statement.

The most complicated step of the whole proof was the version for groups, due to the careful methods that have to be applied to deal with the partial functions. We will insist on this in Section 3.6.5.

When we tried to implement the proof of Lemma 2.2.14, already explored with the symbolic approach in Section 3.5, we tried to apply the same methodology. We introduced the same algebraic structures, with the same representation, and we also continue using the same implementation of homomorphisms. The statement in Isabelle of the lemma is:

lemma *is-reduction*: $[\![$ *diff-group D*; $h \in$ *hom D D*; $h \circ h = (\lambda x.$ *zero D*$)$;
$h \circ$ *diff D* $\circ\, h = h$ $]\!]$
$\Longrightarrow$
$(\!|$ *Up-cc* $=$ *D*,
Down-cc $=$ (*D* $(\!|$ *carrier* $:=$ *Ker D D* $(\lambda x.$ *add D* ((*diff D* $\circ$ *h*) *x*) ((*h* $\circ$ *diff D*) *x*)) $|\!)$),
hom-down $= (\lambda x.$ *add D* ((*id*) *x*) (*a-inv D* (*add D* ((*diff D* $\circ$ *h*) *x*) ((*h* $\circ$ *diff D*)*x*)))),
hom-up $= (\lambda x.$ *id x*$)$,
hom-oper $= h$ $|\!)$ $\in$ *Reduction*

As it can be observed, problems already appear when trying to give the statement in Isabelle, which is hardly readable. In mathematical notation, it can be expressed as $(\mathrm{id}_D - p, \mathrm{inc}_{\ker p}, h)$ will be a reduction from (D, d_D) to $(\ker p, d_D)$ provided that (D, d_D) is a differential group, h an endomorphism of the abelian group D satisfying $hh = 0_D$ and $hd_Dh = h$ and p is $d_Dh + hd_D$.

The main difficulty to express the statement of the lemma in Isabelle in a readable way is due to the way combinations of homomorphisms must be defined. In the statement of Lemma 2.2.11 only composition of functions, an operation already defined in Isabelle libraries, was used. Now, addition of endomorphisms as defined in Section 3.6.3 is also introduced. From it, also the inverse *a-inv* is defined. There are no auxiliary lemmas proved about these operations. Here, two solutions can be proposed. The first would be to state and prove properties that can ease the proofs about *add*, *a-inv*, and composition, $\circ$. Second, we can try to prove a more general result, stating that endomorphisms with these operations (*add* and $\circ$) satisfy the definition of a ring. Then, all previous knowledge in the Isabelle library about rings will be available for working with endomorphisms (and these operations). This second option is based on reusability, and will be shown in the morphism based approach, in Section 3.7.

When we tried to prove the previous statement, expressions grew very quickly and goals split into an exponential number of subgoals. Moreover, most of the goals were related to the definition of the new homomorphisms and endomorphisms through lambda expressions. These new defined functions had to be proved to be homomorphisms or endomorphisms in order the reductions to be well defined.

First, in the following section, we will further explain the advantages and drawbacks of this set theoretic approach, comparing it to the symbolic approach.

3.6.5 Discussion

The symbolic approach exposed in Section 3.5, provided the user with a high level of abstraction in relation to the concrete implementations in the theorem prover; due to this degree of abstraction, it was not precise enough to express and prove some lemmas. With the set theoretic approach, we have presented a framework where the algebraic structures have been represented by records and homomorphisms have been represented as functions. The degree of expressiveness has been augmented. Actually, any of the lemmas we proposed in Section 2.2.2 can be expressed (apart from the graded structures) inside of this setting, this was not possible in the symbolic setting. On the contrary, some improvements can be obtained by changing the way homomorphisms are represented and managed.

There are two main components in this set theoretic framework. One is the algebraic structures involved (groups, abelian groups and differential groups) and the other one is the homomorphisms between these structures. Algebraic structures have been implemented over extensible record types; this has allowed us to use record subtyping and parametric polymorphism, and thus algebraic structures have been derived using inheritance. We have taken advantage of this feature in the proofs of the lemmas; first we have proved a version of Lemma 2.2.11 for groups, then we have used it for proving the same lemma for abelian groups, and finally we have proved the lemma for differential groups, always introducing the existing versions in the new proofs. The model is satisfactory and can be even compared with the Kenzo model, where algebraic structures are also implemented as records.

In the definition of homomorphisms, again a type declaration and a list of axioms satisfied had to be assigned. They were assigned a functional type between generic types in Isabelle. As far as the carriers of algebraic structures are represented through sets over these types, some axioms had to be added ensuring that functions would assign to each element of the source carrier set an element of the target carrier set.

Since in Isabelle equality (=) is total on types, problems arise when comparing homomorphisms (or functions between sets, in general). A way of getting out of this situation is to use the Isabelle constant *arbitrary*, which denotes (for every type) an arbitrary but unknown value. Doing so is sound, because types are inhabited. Functions between sets can be simulated by mapping any element outside of their source set to *arbitrary*.

This is the solution we adopted in this set theoretic approach to be able to use Isabelle equality (=) to compare functions between sets. In our Isabelle definition of isomorphic algebraic structures, we had to use functions which out of their domain of definition were assigned the *arbitrary* value, in order to be able to compare them with the identities. These functions are known in Isabelle as *restrict* functions, and we made use of them both for the isomorphism definition and also inside of the proofs, in order to declare the homomorphisms defining the required isomorphism.

As a consequence of this implementation of function between sets, every function in our model of representation can have several representations in our carrier of representation (i.e., in terms of Isabelle functions). For instance, for any set A, the Isabelle function $\lambda x.x$ represents the identity, but also the Isabelle function $\lambda x.$ *(if $x \in A$ then x else arbitrary)* represents it. This second option is the one we used it in our definition of isomorphism.

From a theoretical point of view, several solutions can be pointed out (for a detailed enumeration, see, for instance [Müller and Slind, 1997]). Here we state three of them, the ones we consider more appropriate to our setting:

1. Equality could be modified to deal with partiality. This solution would imply the definition of a new concept of equality for partial functions depending on three elements, which type declaration and definition would be:

   $$=_{partial\_functions}{::}\ [{'a\ set}, {'a} => {'b}, {'a} => {'b}] => bool$$
   $$=_{partial\_functions}\ A\ f\ g \equiv \forall x \in A. f(x) = g(x)$$

   Modifying the equality by changing the HOL implementation in a system like Isabelle is feasible, but it is hard to guess how this would affect to the degree of mechanization of the system.

2. The one adopted by Isabelle. Isabelle has already defined a concept called *extensional* functions, related to the idea of *restrict* functions, which produces a representative for each set of functions representing the same partial function. The *extensional* functions are defined as:

   constdefs
   extensional :: *'a set => ('a => 'b) set*
   extensional A == $\{f.\ \forall\, x.\ x \notin A$ *-->* $f(x) =$ *arbitrary*$\}$

   An *extensional* function can be understood as a conditional function $\lambda x.$ *(if* $x \in A$ *then* x *else arbitrary)*; therefore, when several *extensional* functions are operated (for instance, combinations of homomorphisms), the number of goals appearing grows exponentially (2^n goals being n the number of homomorphisms, when the definition of *restrict* is unfolded).

   The feasibility of defining an algebraic structure with the homomorphisms inside of the *extensional* functions will be studied in Section 3.7.

3. The use of explicit quotient classes; this solution does not force to change the Isabelle/HOL equality (=) and it produces a similar result to the one obtained using the *extensional* functions. In addition to this, the quotient classes that can be implemented in Isabelle and functions between algebraic structures in mathematics can be easily coupled, while this relation turns to be obscure when introducing an *arbitrary* element, which has no clear meaning in the mathematical setting.

   The ternary relation (defined for every two functions and a set) producing the equivalence classes is given by R_A where

   $$f R_A g \iff \forall x \in A. fx = gx$$

   This relation is reflexive, symmetric and transitive and therefore the equivalence classes for every f in a set A are defined by the following sets: $\{g|\ (\forall x \in A). g(x) = f(x)\}$. Some hints to define equivalence classes in Isabelle can be found in [Paulson, 2004]; how far can the equivalence classes be applied for concrete problems (how to define operations over terms of the equivalence classes) could be of interest.

Depending on this representation of homomorphisms, some functionalities could be added also to improve the mechanization of proofs. We have pointed out the difficulties of proving lemmas about functions in Isabelle, and more concretely, the lack of previous lemmas about functions defined over functions. Here we briefly describe a possible solution to this problem that will be applied in the morphism based approach. In the symbolic approach we emphasized the facilities available in Isabelle for working inside of algebraic structures. Several facts are already proved and even some helpful tactics, such as the already used *algebra* tactic, are implemented. Therefore, if we are able to prove that the set of endomorphisms with the given operations (composition and addition), satisfy the axioms of some algebraic structure, we will be able to instantiate the carrier of the algebraic structure with the set of endomorphisms, and the operations of the algebraic structures with the composition and the addition. Then, we will be able to apply the existing facts about this algebraic structure to the set of endomorphisms, recovering the equational reasoning we used in the symbolic approach, and, moreover, we will be able to also use the properties of the endomorphisms as functions. The same ideas can be applied to homomorphisms.

We will apply these ideas in the morphism based approach.

3.7 The morphism based approach

In this section some improvements are introduced that will be useful to enhance the framework given in the set theoretic approach with some of the features obtained in the symbolic approach. The symbolic approach produced an environment where the representation of both algebraic structures and homomorphisms was a bit distant from their mathematical ones, but with great capacity to reason about homomorphisms equationally. On the other hand, the set theoretic approach improved the representation of algebraic structures and homomorphisms, and therefore increased the expressiveness, but showed some limitations when we tried to implement proofs.

In this new approach we join together the characteristics of both previous frameworks, as well as some other tools improving performance when working with homomorphisms. With this setting, we will be able to give a complete implementation of the proof of Lemma 2.2.14. Moreover, the general ideas proposed can be applied to all six lemmas in Section 2.2.2, and could be also useful in problems requiring the handle of endomorphisms, homomorphisms and their properties.

3.7.1 The algebraic structures

Lemma 2.2.14 has been already exposed and faced in the symbolic approach, and the algebraic structures present on it enumerated, so here just a brief description is being done: in the statement of the lemma appear explicitly the differential group (D, d_D), the differential subgroup $(\ker p, d_D)$, the abelian group endomorphisms h, 0_D, the abelian group homomorphism $d_D h + h d_D$, and the abelian group homomorphisms $\mathrm{id}_{\mathrm{End}(D)}$ and $\mathrm{inc}_{\ker p}$, while implicitly appear, at least, the rings $\mathrm{End}(D)$, $\mathrm{End}(\ker p)$, the ring $\mathrm{End}((D, d_D))$, the abelian groups $\mathrm{Hom}(D, \ker p)$, $\mathrm{Hom}(\ker p, D)$, the abelian groups $\mathrm{Hom}((D, d_D), (\ker p, d_D))$, $\mathrm{Hom}((\ker p, d_D), (D, d_D))$, and the ringoid formed taking the abelian group and differential group endomorphisms and homomorphisms of these structures.

In the Isabelle proof of Lemma 2.2.14, the following algebraic structures will be implemented:

- The differential groups (D, d_D) and $(\ker p, d_D)$ (as in the set theoretic approach).
- The endomorphisms of the differential group (D, d_D), d_D and $\mathrm{id}_{\mathrm{End}(D)}$, and the endomorphism h of the abelian group D, as well as combinations of them (for instance, $p = d_D h + h d_D$) (as in the set theoretic approach, although they will have a different representation).
- The *ring* of endomorphisms $\mathrm{End}((D, d_D))$ as well as some lemmas that will allow us to establish equivalences among the elements of the ring $\mathrm{End}((D, d_D))$ with elements of the algebraic structures $\mathrm{Hom}((\ker p, d_D), (D, d_D))$, $\mathrm{Hom}((D, d_D), (\ker p, d_D))$ and $\mathrm{End}((\ker p, d_D))$, by modifying the source or target of combinations of endomorphisms under determined premises (in contrast with the set theoretic approach, where endomorphisms and homomorphisms were elements of a set).

Some changes will be also introduced in the representation of endomorphisms and homomorphisms with respect to the set theoretic approach, that will allow us to prove that the endomorphisms of a differential group, with the adequate operations, satisfy the axioms of a ring. As a consequence of this, we will be able to apply the previous knowledge proved in Isabelle for this algebraic structure (in this particular case, rings), for our set of endomorphisms.

In addition to this, the representation of homomorphisms will store information about their source and target algebraic structures. This will increase the capabilities to deal with partiality in a setting where elements of $\mathrm{End}((D, d_D))$ have to be expressed, but also elements of $\mathrm{Hom}((\ker p, d_D), (D, d_D))$, $\mathrm{Hom}((D, d_D), (\ker p, d_D))$ and $\mathrm{End}((\ker p, d_D))$ appear.

3.7.2 Representation of the algebraic structures

Implementation of algebraic structures does not suffer modifications in this new approach. Differential groups are implemented as sets of terms of a generic type over which the operations of the algebraic structure are defined and some axioms are imposed. The function of abstraction for differential groups is the same as in the set theoretic approach.

The representation chosen in the set theoretic approach by extensible records together with axioms showed to be precise, since it fits in the theoretical concept of signature of algebraic structures, and moreover, it is adequate from the point of view of code optimization, thanks to structural subtyping.

In this approach, we will make use again of the Isabelle implementation of the *ring* algebraic structure to implement the algebraic structure formed by the set of elements $\mathrm{End}((D, d_D))$. The representation chosen for rings will be the one given in Section 1.3.2 and already used in the symbolic approach exposed in Section 3.5, where it was also used to describe the algebraic structure formed by endomorphisms. Here it will be useful in order to obtain the degree of automation reached there, although here the ring will be instantiated with functional objects representing endomorphisms, instead of being declared as containing terms of generic type (i.e., the semantical content of the framework will be increased).

3.7.3 Homomorphisms between algebraic structures

In this section we describe the requirements we had to encode in Isabelle in order to be able to define a ring on the set of endomorphisms over a differential group (D, d_D). In order to obtain this algebraic structure, first we must define which objects (defined through a type and a collection of axioms) will be chosen as endomorphisms, then the operators that form the ring must be selected, and finally the ring axioms must be proved with the defined operators and endomorphisms. One of the problems pointed out in the set theoretic approach was that the abstraction function for homomorphisms and endomorphisms was not injective. One homomorphism from the model of representation had several preimages in the Isabelle representation (following the terminology introduced in Section 3.4, the representation was not faithful).

Here we propose a possible solution in Isabelle, which will be shown to be useful to implement proofs improving the presentation obtained in the set theoretic approach. Different solutions can be proposed (some of them have been already mentioned in Section 3.6.5). The one allowing to maintain the degree of mechanization obtained in our proofs would be the definition of the *restrict* operator. The *restrict* operator, as previously explained, takes a function f and a set A as arguments, and returns the function defined $\lambda x.$ *(if* $x \in A$ *then* x *else arbitrary)*. This is the solution requiring a smaller number of modifications, since this method neither needs to change the equality of the system, nor to deal with equivalence classes; moreover, what is obtained by working with the *restrict* result of every total function is a kind of representative of the sets of equivalence classes produced by the relation $f R_A g \iff \forall x \in A. f(x) = g(x)$.

At first sight, the only drawback of the *restrict* functions is that they have as result an *arbitrary* element when applied to an element out of their source set. This *arbitrary* element could be misleading when trying to identify the functions we have defined in Isabelle with mathematical homomorphisms, i.e., when defining the abstraction function allowing us to identify the Isabelle terms with mathematical entities.

But in addition to this, another fact can be pointed out that made us change our mind to a slightly different representation of homomorphisms. In the ring of endomorphisms we would like to define in order to work with endomorphisms applying equational reasoning, the multiplicative operation would be the usual composition between functions, represented in Isabelle with the symbol $\circ$. This operation is not closed under functional objects defined using the *restrict* operator. In order to show it, let us first recover the concept of *extensional* functions over a given set. A function f is said to be *extensional* for a set A (according to the definition in the Isabelle library *Funcset.thy*) if and only if:

$$f = \mathit{restrict}\ f\ A$$

Therefore, it is equivalent to work with the result of applying *restrict* to a function than to do it with *extensional* ones (i.e., every *extensional* function over a set can be expressed as a function in its *restrict* form, and the *restrict* result of a function is also an *extensional* function). If we pretend to prove that the set of *extensional* functions is adequate to represent endomorphisms, one of the first properties that must be proved is that this set is closed under composition (which would correspond to the *mult* operation of our ring). The problem now is that the composition of two *extensional* functions over a given set A does not yield an *extensional* function. Let us consider two such functions $\lambda x.$ *(if* $x \in A$ *then* $f(x)$ *else arbitrary)* and $\lambda x.$ *(if* $x \in A$ *then* $g(x)$ *else arbitrary).* As far as we are considering endomorphisms over the set A, it can be assumed that $f(x)$ will be an element of A for $x \in A$, an so is $g(f(x))$. Therefore, for $x \in A$, the composition behaves as expected. The problem appears when we consider an element x such that $x \notin A$ (as it has been already mentioned, sets are partial over types). Then, the result of applying $\lambda x.$ *(if* $x \in A$ *then* $f(x)$ *else arbitrary)* (the same could be done with the other function) to x will be *arbitrary.* And now, when applying $\lambda x.$ *(if* $x \in A$ *then* $g(x)$ *else arbitrary)* to *arbitrary,* as far as *arbitrary* has no semantical content, due to the law of the excluded middle, cases *arbitrary* $\in A$ and *arbitrary* $\notin A$ have to be considered. The second case behaves also correctly (elements out of A are mapped to *arbitrary* by *extensional* functions, and then composition in this case would be closed). On the contrary, in the case where *arbitrary* is considered in A, the result of the composition is $g(\mathit{arbitrary})$, from which no semantical information is available (we do not even know if it is equal to *arbitrary* or not). Therefore, composition of *extensional* functions is not closed.

A way for getting out of this situation consists in defining a new composition (named *compose*). This new definition is also found in the Isabelle library, and satisfies that the composition of two *extensional* functions is an *extensional* function:

constdefs
  compose :: [*'a set*, *'b* => *'c*, *'a* => *'b*] => (*'a* => *'c*)
  compose A g f == *restrict (g* $\circ$ *f) A*

Applying this solution would have some consequences. First, from the point of view of code optimization, in the Isabelle library very few facts are available about the *compose* operator, which would decrease the efficiency of the automatized tactics. From the point of view of the representation of the proofs obtained, this solution implies turning the composition definition into a ternary function (the set over which composition our endomorphisms are defined and both functions). Moreover, the *restrict* operator introduces the *arbitrary* constant in the definition of the composition. This constant should be supplied a mathematical meaning through the process of abstraction in the model of the representation (i.e., the set A over which endomorphisms are defined).

We preferred a second solution, which allows us to use $\circ$ as the composition in the ring of endomorphisms we pretend to define, and which also simplifies the abstraction process. Instead of considering as endomorphisms the *extensional* result of the total functions satisfying the Isabelle definition of endomorphisms, where *arbitrary* was the constant chosen from the codomain set, we will choose another distinguished element of the codomain set. The only constant of the codomain set which is bound to exist is the unity element of the codomain, *one*, since in our proofs, at least, homomorphisms between monoids are considered. Then, the function which turns a homomorphism into its unique representative will be:

constdefs
  completion :: [('a, 'c) *monoid-scheme*, ('b, 'd) *monoid-scheme*, ('a => 'b)] => ('a => 'b)
  completion G G' f == (λx. *if* $x \in$ *carrier G then f x else one G'*)

Now, it can be also defined the set of all the *completion* functions between two monoids G and G':

constdefs
  completion-fun2 :: [('a, 'c) *monoid-scheme*, ('b, 'd) *monoid-scheme*] => ('a => 'b) *set*
  completion-fun2 G G' == {*f*. ∃ *g*. *f* = *completion G G' g*}

The following facts can be remarked about this definition:

- We named it *completion*, based on the mathematical idea behind this concept of making complete a representation of some partially defined entity; all the elements out of the domain are mapped to a distinguished element of the codomain structure (*one* in our case).
- Given G, G' monoids, every function in the set *completion-fun2* of the monoids G and G' can be identified with a function of the set *extensional* over the set *carrier* G. The correspondence is defined by applying the elements of the form

  (λx. *if* $x \in$ *carrier G then f*(x) *else one G'*)

  to the *extensional* function

$(\lambda x.$ *if* $x \in$ *carrier G then f*(x) *else arbitrary*) .

This characterization of functions has to be added to the implementation of homomorphisms that we already had. This is what we get by defining the *hom-completion* set, which again will depend on two parameters of type *monoid*, G and G':

constdefs
hom-completion :: [(*'a*, *'c*) *monoid-scheme*, (*'b*, *'d*) *monoid-scheme*] => (*'a* => *'b*)*set*
hom-completion G G' == {*h. h* ∈ *completion-fun2 G G'* & *h* ∈ *hom G G'*}

Now, we can consider again the representation process. Our model of representation would be the set of functions ${G'}^G$ (G and G' denote now the carrier sets of the algebraic structures G and G'), where G and G' are known to have a distinguished element, let us say *one G* and *one G'*. The domain of the representation is similar to the one obtained in the set theoretic approach, i.e., the type of functions from the type of G into the type of G'. The carrier of the representation now changes, being formed by the functions such that every element in G is mapped to an element in G', and every element out of G is mapped to the element *one G'*. The equality of the representation, which is the equality in Isabelle and thus is total on types, allows us to identify any two objects equal for every element of G, in contrast with the set theoretic approach where this can not be done (there, different Isabelle functions represented the same homomorphism). Then, the abstraction function will be now injective (i.e., the representation is faithful), and each function of the model of the representation will have, at maximum, one preimage in the carrier of the representation. Now we will be able to, maintaining the Isabelle definition of equality for our model, and with the obtained uniqueness together with some suitable operations, define some concrete algebraic structures (for instance, a ring of endomorphisms, where the unit is uniquely determined).

The two reasons that made us avoiding the use of *extensional* functions have been overcome. The first one was that the *extensional* functions with the *arbitrary* constant did not have a clear translation into mathematical terms. The situation is fixed now by using the constant *one* of the target structures. The second reason was that the *extensional* functions were not even closed under usual composition. On the contrary, *hom-completion* functions can be shown to be closed in Isabelle through the following statement:

lemma *hom-completion-comp-is-closed*: **assumes** *group G* **and** *group G'* **and** *group G''*
and *f* ∈ *hom-completion G G'* **and** *g* ∈ *hom-completion G' G''*
shows *g* ∘ *f* ∈ *hom-completion G G''*

Its proof in Isabelle took us less than 20 lines of code.

The set of functions *hom-completion* $G\,G$ can be proved to be a monoid for G a monoid, with the composition ∘ as multiplication. The set *hom-completion* $G\,G'$ can be proved to be an abelian group considering like addition the inherited operation from G', with G and G' abelian groups; we will face these proofs later. First, we briefly explain the reasons that led us to introduce the domain and codomain of homomorphisms explicitly (as fields of a record) in the homomorphisms' representation of this framework.

In the statement of Lemma 2.2.14, two differential groups appear[8]. The first one is (D, d_D), and the second one is $(\ker p, d_D)$, where p is an endomorphism of the differential group (D, d_D); as it has been proved in Proposition 2.2.12, $(\ker p, d_D)$ is a differential subgroup of (D, d_D). In order to produce a proof in Isabelle of Lemma 2.2.14 different solutions can be proposed. One of them would be to apply the tools introduced in the set theoretic approach. We already tried it with an unsatisfactory result. Another one would be to take advantage of the properties of the algebraic structures $\operatorname{End}((D, d_D))$, $\operatorname{Hom}((\ker p, d_D), (D, d_D))$, $\operatorname{Hom}((D, d_D), (\ker p, d_D))$ and $\operatorname{End}((\ker p, d_D))$; nevertheless, this would require to design a representation in Isabelle of all these algebraic structures. A third solution, and the one we explore in this approach, consists in implementing the proofs using only properties of the ring $\operatorname{End}((D, d_D))$, and taking advantage of the relationship between the differential group and its subgroup, implement some tools (or lemmas) that allow us to identify equalities between elements of this ring with equalities between terms of the algebraic structures $\operatorname{Hom}((\ker p, d_D), (D, d_D))$, $\operatorname{Hom}((D, d_D), (\ker p, d_D))$ and $\operatorname{End}((\ker p, d_D))$.

The following reasons were considered to choose this solution:

1. This representation allows us to prove equalities where elements of the sets $\operatorname{End}((D, d_D))$, $\operatorname{Hom}((\ker p, d_D), (D, d_D))$, $\operatorname{Hom}((D, d_D), (\ker p, d_D))$ and $\operatorname{End}((\ker p, d_D))$ are involved just by using elements of the set $\operatorname{End}((D, d_D))$. The methodology is as follows. First, the expression to be proved is defined in terms of elements of $\operatorname{End}((D, d_D))$, just by changing the source or the target of the endomorphisms and homomorphisms; then, computations are carried out taking advantage of the properties of the ring $\operatorname{End}((D, d_D))$, in what we called in Section 3.5 an *equational way*, making use of the automatized tactics inside of rings available in Isabelle; finally, under some determined premises, the result obtained in the form of an equality between elements of $\operatorname{End}((D, d_D))$ can be used to derive an equality where elements of $\operatorname{End}((D, d_D))$, $\operatorname{Hom}((\ker p, d_D), (D, d_D))$, $\operatorname{Hom}((D, d_D), (\ker p, d_D))$ and $\operatorname{End}((\ker p, d_D))$ can be involved.

2. The lemmas that allow us to modify the domain or codomain of endomorphisms, that we will introduce later, are quite simple to state and also to prove, both in mathematical terms and also in Isabelle. Actually, they will be exposed in the form of one single lemma (Lemma 3.7.1), that solves all the possible situations.

The representation for homomorphisms with the given features will be as follows (as usual in Isabelle definitions, first a type is given, and later the set of rules satisfied is assigned):

[8]As pointed out in Section 1.3.2, when dealing with differential groups, the operations *mult*, *one* and *m-inv* become *add*, *zero* and *a-inv*.

```
record ('a, 'b, 'c, 'd) group-mrp-type =
  src           ::  ('a, 'c) diff-group-scheme
  trg           ::  ('b, 'd) diff-group-scheme
  morph         ::  'a ⇒ 'b
  src-comm-gr   ::  ('a, 'c)
  trg-comm-gr   ::  ('b, 'd) diff-group-scheme
```

The record type has been named *group-mrp-type* due to the similarity of the previous declaration with the representation of morphisms in Category Theory (where the source and target structures are usually made explicit). The type is defined through a record with five fields. One of them, *morph*, contains the functional information, in the form of a *hom-completion* object. In the record fields *src-comm-gr* and *trg-comm-gr* will be stored, respectively, information about the greatest differential group that can be the source of the homomorphism, and the greatest differential group that can be the target of the homomorphism (in our setting, the differential groups appearing are (D, d_D) and differential subgroups of it). Then, the differential groups where the homomorphism is precisely defined at each concrete step of a proof are kept in the fields *src* and *trg*. This definition ensures that we can modify the domain and codomain of a homomorphism (fields *src* and *trg*) when some properties are satisfied, maintaining the information about where it is really well defined (stored in fields *src-comm-gr* and *trg-comm-gr*, which are constant); this definition will allow us to change the *src* and *trg* fields in such a way that the definition of the morphism is always correct.

Following these ideas, the axioms that must be satisfied by a tuple with type *group-mrp-type* in order to be considered a homomorphism in our setting are:

```
locale group-mrp2 =        struct MRP +
  assumes src-diff-group:  diff-group (src MRP)
  and src-comm-group:      diff-group (src-comm-gr MRP)
  and src-subgroup:        sub-diff-group2 (src MRP) (src-comm-gr MRP)
  and trg-diff-group:      diff-group (trg MRP)
  and trg-comm-group:      diff-group (trg-comm-gr MRP)
  and trg-subgroup:        sub-diff-group2 (trg MRP) (trg-comm-gr MRP)
  and image-in-trg:        (morph MRP) ‘ (carrier (src MRP)) ⊆ carrier (trg MRP)
  and morph-hom:           morph MRP ∈
                           hom-completion (src-comm-gr MRP) (trg-comm-gr MRP)
```

The most relevant ones are *src-subgroup*, which ensures that the source of a tuple, *src*, will be always a differential subgroup of the fixed differential group *src-comm-gr*, where the homomorphism is known to be well-defined, *image-in-trg*, establishing that the image of the *src* differential group will be always contained in the *trg* differential group, and *morph-hom*, establishing that the *morph* field of a tuple contains a *hom-completion* object between the differential groups *src-comm-gr* and *trg-comm-gr*.

An Isabelle definition of differential subgroups, *sub-diff-group2*, will be also supplied in this setting (it has been already used in definition *group-mrp2*). The Isabelle definitions of submagma, subgroup or abelian subgroup are made in the form of a subset satisfying some rules. Algebraic structures are expressed in Isabelle as records, and thus algebraic substructures do not satisfy the Isabelle definitions of the algebraic structures by themselves. The intensive use of the substructure relation already expressed in the Isabelle definition *group-mrp2* made us introduce a definition of differential subgroup such that a differential subgroup is also a differential group by itself (in the sense that it contains a carrier and also the corresponding operations).

The following Isabelle definition ensures that, given D a differential group and D' a differential subgroup of it, D' inherits all its operations of the differential group D, and moreover, is a differential group by itself:

locale *sub-diff-group2* = *diff-group* D + *diff-group* D' +
  assumes *carrier-cont*: *carrier* $D \subseteq$ *carrier* D'
  and *equal-op*: $D = (\!|$ *carrier* = *carrier* D, *add* = *add* D', *zero* = *zero* D', *diff* = *diff* D', ... = *diff-group.more* D' $|\!)$

With the new definition of endomorphisms, now the operations for working with them have to be defined. The composition (that will be the multiplicative operation in the ring of endomorphisms) of two *group-mrp2* objects f and g is defined in Isabelle as expected:

constdefs
group-mrp-comp :: [('b, 'c, 'e, 'f) *group-mrp-type*, ('a, 'b, 'd, 'e) *group-mrp-type*]
  => ('a, 'c, 'd, 'f) *group-mrp-type*
group-mrp-comp g f == $(\!|$ *src* = *src* f, *trg* = *trg* g, *morph* = (*morph* g) $\circ$ (*morph* f), *src-comm-gr* = *src-comm-gr* f, *trg-comm-gr* = *trg-comm-gr* $g$$|\!)$

The functional part of the definition is based on the usual composition in Isabelle, $\circ$. As it was previously exposed, the representation of endomorphisms through *completion* functions allows us to use the usual composition of functions. This composition will be denoted, when no confusion arises, as the usual composition (otherwise, we will refer to it as *group-mrp-comp*). With this definition of composition, some lemmas are proved ensuring that the composition of two *group-mrp2* objects f and g is again a *group-mrp2* object whenever the *trg* and the *trg-comm-gr* of f are equal to the *src* and the *src-comm-gr* of g, and also when the *trg* of f is equal to the *src* of g, and the *trg-comm-gr* of f is a differential subgroup of the *src-comm-gr* of g. More situations where the composition is closed (i.e., produces a *group-mrp2*) object can be proposed, but they are not used in the proofs we are facing.

Once that composition has been defined in a natural way for homomorphisms, we have to prove that the endomorphisms, with the suitable operations, satisfy the axioms of a ring. The carrier set will be formed by *group-mrp2* objects, and the multiplicative operation, as already shown, will be *group-mrp-comp*, which is known to be closed.

The definition of the set of homomorphisms between two algebraic structures resembles the one already given in Section 3.6.3; here four differential groups must be considered, following the definition of *group-mrp2*:

```
constdefs
group-mrp-set  ::   [ ('a, 'c) diff-group-scheme, ('b, 'd) diff-group-scheme,
                      ('a, 'c) diff-group-scheme, ('b, 'd) diff-group-scheme]
                      => (('a, 'b, 'c, 'd) group-mrp-type) set
group-mrp-set A B G1 G2   ==   {morph.
                                  src morph = A & trg morph = B &
                                  src-comm-gr morph = G1 &
                                  trg-comm-gr morph = G2 & group-mrp2 morph}
```

Every tuple belonging to this set must satisfy the definition given of *group-mrp2*. Now, the set of endomorphisms of a given differential group D can be seen as a particular case of the previous definition, when we consider $A = B = G1 = G2 = D$. The set *group-mrp-set* $D\,D\,D\,D$ represents the set of endomorphisms over a generic differential group D. Now we will focus our attention on the properties verified by the set *group-mrp-set* $D\,D\,D\,D$ and later we will see how the *src* and *trg* fields can be changed under some determined premises.

Considering *group-mrp-set* $D\,D\,D\,D$ like carrier set, the properties of a ring must be verified. The multiplicative operation will be *group-mrp-comp*. The definition of a ring is done in Isabelle like the union of a monoid and an abelian group, proving also that the multiplicative operation is distributive with respect to the additive one. Following this definition, we first prove that *group-mrp-set* $D\,D\,D\,D$ with *group-mrp-comp* as multiplication defines a monoid. The statement in Isabelle of the lemma would be:

```
lemma group-mrp-set-monoid:
  includes diff-group D
  shows monoid
  (|carrier = group-mrp-set D D D D,
   mult = group-mrp-comp,
   one = (|src = D, trg = D, morph = (λx. if x ∈ carrier D then id x else zero D),
           src-comm-gr = D, trg-comm-gr = D|) |)
```

Its proof requires over 40 lines of Isabelle code (once we have proved that composition is closed, which was a bit more complicated). The only condition imposed is that D must be a differential group. The unit for the multiplication corresponds to the *completion* version of the identity function:

```
(λx. if x ∈ carrier D then id x else zero D)
```

Now, the additive operation, based on the additive operation of the underlying differential group, is given by (for A and B differential groups):

```
constdefs
  group-mrp-mult  ::   [ ('a, 'a, 'b, 'b) group-mrp-type, ('a, 'a, 'b, 'b) group-mrp-type]
                        => ('a, 'a, 'b, 'b) group-mrp-type
```

```
group-mrp-mult A B   ==   (| src = src A, trg = trg A,
                              morph = (λx. if x ∈ carrier (src A)
                                           then (add (trg A) (morph A x)
                                                             (morph B x))
                                           else (zero (trg A))),
                              src-comm-gr = src-comm-gr A,
                              trg-comm-gr = trg-comm-gr A|)
```

This definition is more generic than the one we need (where $A = B = D$, with D a differential group), being valid for objects of type *group-mrp* where the *src* and the *trg* are not the same.

Now, we fix again a differential group D, and taking as carrier set *group-mrp-set* $D\,D\,D\,D$, and *group-mrp-mult* as addition, the axioms of abelian group have to be proved:

```
lemma group-mrp-set-comm-group:
  includes diff-group D
  shows comm-group  (|  carrier = group-mrp-set D D D D, mult = group-mrp-mult,
                        one = (|src = D, trg = D,
                                morph = (λx. if x ∈ carrier D
                                             then zero D else zero D),
                                src-comm-gr = D, trg-comm-gr = D|) |)
```

This proof in Isabelle required over 200 lines. As it has been explained, when defining the ring of endomorphisms, the *one* field here will be the *zero* of the ring, and the *mult* will become the *add* operation for the ring.

Making use of the Isabelle lemmas *group-mrp-set-monoid* and *group-mrp-set-comm-group*, and proving also the distributivity of *group-mrp-comp* with respect to *group-mrp-mult*, we finally obtain that the following record satisfies the axioms of the Isabelle *ring* definition:

```
(| carrier   =   group-mrp-set D D D D,
   mult      =   group-mrp-comp,
   one       =   (|src = D, trg = D,
                   morph = (λx. if x ∈ carrier D then id x else zero D),
                   src-comm-gr = D, trg-comm-gr = D|),
   zero      =   (|src = D, trg = D,
                   morph = (λx. if x ∈ carrier D then zero D else zero D),
                   src-comm-gr = D, trg-comm-gr = D|),
   add       =   group-mrp-mult |)
```

With this result, now we can apply all the results (or lemmas) already proved in the Isabelle library for rings to the set of endomorphisms over a differential group with the given operations. In other words, we have defined a new level of abstraction for our proofs. Now, working with endomorphisms can be done in an *equational way*, as in the symbolic approach. In addition to this, some advantages can be highlighted with respect to the representation chosen in the symbolic approach:

- The representation of endomorphisms is done now through records which keep a functional field, whereas in the symbolic approach endomorphisms were just represented as elements of a generic type.
- The algebraic structures D and $\ker p$ have now an explicit representation, implemented through records, instead of being just implicitly represented, as they were in the symbolic approach.
- Representing the endomorphisms with a functional field allows us to prove facts considering the concrete elements of the endomorphisms' domain and codomain, and therefore, to improve the reasoning tools available in the symbolic approach.

With respect to the set theoretic approach, we have obtained the capacity to deal with endomorphisms as objects of an algebraic structure, a ring, with concrete operators, while in the set theoretic approach every operation had to be made working with endomorphisms as functions, relying strongly in the implementation details, and with a very low level of abstraction.

Then, the advantages of both approaches have been combined in this approach. We now illustrate with a very simple lemma the differences, both at the level of notation and also of proof mechanization, between the set theoretic approach and the morphism based approach.

Once we have proved that the endomorphisms with the corresponding operations form a ring, and in order to improve notation, we can define a *locale* context, where two variables D and R are fixed, where D is a differential group and R the ring of endomorphisms (over D). Now, and thanks to the previous lemma *group-mrp-set-ring*, we link R to the set of endomorphisms over D with the given operations:

```
locale group-mrp-set-ring = diff-group D + ring R +
 assumes R ≡ (| carrier  = group-mrp-set D D D D,
                mult     = group-mrp-comp,
                one      = (|src = D, trg = D,
                              morph = (λx. if x ∈ carrier D then id x else zero D),
                              src-comm-gr = D, trg-comm-gr = D|),
                zero     = (|src = D, trg = D,
                              morph = (λx. if x ∈ carrier D then zero D else zero D),
                              src-comm-gr = D, trg-comm-gr = D|),
                add      = group-mrp-mult |)
```

Now, with the pretty printing symbols defined in Isabelle for rings, and while being inside of the context created by the *locale*, syntax can be greatly improved, allowing, for instance, to invoke the multiplication in R, which is *group-mrp-comp*, like $\otimes_2$, and the addition, *group-mrp-mult*, like $\oplus_2$.

The following lemma inside of this setting can be proved with one tactic, *algebra*, designed to produce simplifications inside of some algebraic structures (as it was done in the symbolic approach):

lemma (**in** *group-mrp-set-ring*) *example-morphism-approach*:
  [| $f \in$ *carrier R*; $h \in$ *carrier R*; $f \otimes_2 h = h$ |] ==> $(\mathbf{1}_2 \ominus_2 f) \otimes_2 h = \mathbf{0}_2$
  by *algebra*

Here $\mathbf{1}_2$ represents the *one* field of the ring R (i.e., the functional object $\lambda x.$ *(if x ∈ carrier D then id x else zero D))*, $\mathbf{0}_2$ represents the *zero* field of R, and also $\otimes_2$ and $\ominus_2$ make reference to these operations as defined in R (i.e., *group-mrp-comp* and the opposite of *group-mrp-mult*). Now it can be observed the difference between this approach and the symbolic one, where **1** was a shortcut of the unity of a generic ring R, and $\otimes$ of a multiplication from which no more information was provided.

The subindexes in the operators ($\otimes_2$, $\oplus_2$, $\mathbf{1}_2$, ...) refer to the number of structure in the context *locale*. Here, subindex 1 would refer to operators in the first structure, i.e. D, and subindex 2 to the operators in the second structure, the ring R. The correspondence is in the order that the variables are fixed when the locale is declared.

The same lemma, with its proof, expressed with the available tools in the set theoretic approach, would be as follows:

lemma *example-set-theoretic-approach*: **assumes** *diff-group D* **and** $f \in$ *hom-completion D D* **and** $h \in$ *hom-completion D D* **and** $f \circ h = h$
  shows (λ *x. add D* ((λ *x. if x* ∈ *carrier D then id x else zero D*)*x*) (*a-inv D* (*f x*))) ∘ *h*
  = (λ *x. if x* ∈ *carrier D then zero D else zero D*)
proof
  fix *x*
  show ((λ*x. add D* (*if x* ∈ *carrier D then id x else zero D*) (*a-inv D* (*f x*))) ∘ *h*) *x* =
    (λ*x. if x* ∈ *carrier D then zero D else zero D*) *x*
  proof (*cases x* ∈ *carrier D*)
    case *True*
    from *prems* **show** *?thesis*
      by (*auto simp add*: *expand-fun-eq hom-completion-def hom-closed*)
        (*intro monoid.Units-r-inv*, *unfold diff-group-def monoid-def group-axioms-def hom-completion-def*, *auto simp add*: *hom-closed*)
  next
    case *False*
    from *prems* **show** *?thesis*
    proof (*cases h x* ∈ *carrier D*, *auto simp add*: *expand-fun-eq*)
      case *True*
      from *prems* **show** *add D* (*h x*) (*a-inv D* (*h x*)) = *zero D*
        by (*intro monoid.Units-r-inv*, *unfold diff-group-def monoid-def group-axioms-def hom-completion-def*, *auto simp add*: *hom-closed*)
    next
      case *False*
      from *prems* **show** *add D* (*zero D*) (*a-inv D* (*h x*)) = *zero D*
        by (*unfold diff-group-def hom-completion-def completion-fun2-def completion-def*, *auto simp add*: *monoid-axioms-def*)

```
    qed
  qed
qed
```

It cannot be argued that the length of the Isabelle proof of *example-set-theoretic-approach* is huge or disappointing, despite the triviality of the lemma, but both in the statement and also in the proof, it can be observed that the design decisions directly affect to the statement and the proof steps, making the proof hard to implement and also to read; these design decisions did not affect directly to the statement and proof of *example-morphism-approach*.

The benefits obtained to express and also to reason with combinations of homomorphisms in this setting, with respect to the set theoretic approach and the default tools available in Isabelle, as well as the difficulties pointed out in Section 3.6.4, should be noticed. The more complicated the combinations appearing in the statement are, the more obscure becomes the statement (and the proof) using the tools provided by the set theoretic approach.

A new feature can be added to this framework, augmenting its usefulness (in the sense that it will allow us to represent elements out of the set of endomorphisms of D, e.g., homomorphisms from D to $\ker p$, or endomorphisms of $\ker p$). By now, we are able only to express, with elements of R, elements of the ring of endomorphisms of a differential group D. Making use of the information stored in the *group-mrp2* record, it will be also possible to express homomorphisms of differential subgroups of D (in our case, the differential subgroup will be $\ker p$). Adding also some special lemmas, equalities between elements of the ring R will be also equivalent to equalities between elements of these differential subgroups.

The lemma allowing us such modifications is the following:

Lemma 3.7.1. *Let f be a differential group homomorphism between the differential groups A and B. Let A' be a differential subgroup of A, B' a differential group, and $\operatorname{im} f(A') \subseteq B'$. Then, we consider the tuple $\langle A', B', f, A, B\rangle$. Similarly, let g be a differential group homomorphism between the differential groups C and D, with C' a differential subgroup of C and D' a differential group such that $\operatorname{im} g(C') \subseteq D'$, and let us consider $\langle C', D', g, C, D\rangle$. Let B' be a differential subgroup of C'. Let $\langle C', D', g, C, D\rangle \circ \langle A', B', f, A, B\rangle$ be equal to $\langle A', D', h, A, D\rangle$. Then, if A'' is a differential subgroup of A', B'' a differential subgroup of C'', such that $\operatorname{im} f(A'') \subseteq B''$, and C'' a differential subgroup of C' such that $\operatorname{im} h(C'')$ contained on D'', with D'' a differential subgroup of D', the following equality $\langle C'', D'', g, C, D\rangle \circ \langle A'', B'', f, A, B\rangle = \langle A'', D'', h, A, D\rangle$ holds, being $\circ$ the operation group-mrp-comp.*

Proof. We consider $x \in A''$; from the premises, we know that $x \in A'$. Therefore, $f(x)$ will be an element of B'', being then the tuple $\langle A'', B'', f, A, B\rangle$ well defined; moreover, B'' is a differential subgroup of C'', and so the composition $\langle C'', D'', g, C, D\rangle \circ \langle A'', B'', f, A, B\rangle$

is also well defined, since $g(f(x))$ will be an element of D''. In addition to this, the equality $\langle C'', D'', g, C, D\rangle \circ \langle A'', B'', f, A, B\rangle = \langle A'', D'', h, A, D\rangle$ holds because for all x in A'', in particular x is in A', and $g(f(x)) = h(x)$. ■

It can be observed that the lemma can be unfolded into various lemmas modifying each of the components (domains and codomains) of the tuples, but we did not consider it necessary.

Applied to Lemma 2.2.14, where the only differential groups appearing in the statement are (D, d_D) and $(\ker p, d_D)$, some equalities between elements of the ring $\mathrm{End}((D, d_D))$ allow to prove equalities where elements of $\mathrm{Hom}((D, d_D), (\ker p, d_D))$, $\mathrm{Hom}((\ker p, d_D), (D, d_D))$ or also of $\mathrm{End}((\ker p, d_D))$ appear.

Now, with the possibility of working with endomorphisms in an equational way, as well as in a *natural* way (i.e., considering the elements of their domain and their properties), and also thanks to this lemma which modifies the domain and codomain of endomorphisms under determined circumstances, we will be able to give a complete implementation of the proof of Lemma 2.2.14.

3.7.4 Lemmas proved

In this section we will describe the implementation of the proof of Lemma 2.2.14 given in this framework; we will leave for Section 3.7.5 the analysis of the possibilities of applying the tools described here to the rest of the lemmas given in Section 2.2.2 and also to other different problems.

In Lemma 2.2.14 a reduction is defined between the differential group (D, d_D) and its differential subgroup $(\ker p, d_D)$, by means of $(\mathrm{id}_{\mathrm{End}(D)} - p, \mathrm{inc}_{\ker p}, h)$ being $p = d_D h + h d_D$. According to the definition of reduction, the following properties must be satisfied:

1. $(\mathrm{id}_{\mathrm{End}(D)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\ker p}$;
2. $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(D)} - p) + d_D h + h d_D = \mathrm{id}_{\mathrm{End}(D)}$;
3. $(\mathrm{id}_{\mathrm{End}(D)} - p)h = 0_{\mathrm{Hom}(D,\ker p)}$;
4. $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p, D)}$;
5. $hh = 0_{\mathrm{End}(D)}$.

In the symbolic approach, despite the low level of detail introduced, we were able to give a proof of two of these five conditions, $(\mathrm{id}_{\mathrm{End}(D)} - p)h = 0_{\mathrm{Hom}(D,\ker p)}$ and also the trivial one $hh = 0_{\mathrm{End}(D)}$. On the other hand, we exposed the problems found even to express, and consequently to prove, the remaining ones.

In the present approach, mainly due to the representation chosen for endomorphisms, the size of the statements is greater than in the previous approaches, which clearly goes against readability. On the other hand, the degree of expressiveness and also of automation has been largely enhanced.

We will continue using the *locale group-mrp-set-ring* where D was declared to be a differential group, and R the ring of the endomorphisms (as tuples) with the corresponding operations.

The property $hh = 0_{\mathrm{End}(D)}$ can be proved in a single step, simplifying it with the premises. The statement of property $(\mathrm{id}_{\mathrm{End}(D)} - p)h = 0_{\mathrm{Hom}(D, \ker p)}$ has grown in relation to the statement of the same property in the set theoretic approach, being now:

lemma (**in** *group-mrp-set-ring*) *third-condition*:
 assumes *diff-morph*: *diff-morph* = (| *src* = *D*, *trg* = *D*, *morph* = *differ*,
 src-comm-gr = *D*, *trg-comm-gr* = *D* |)
 and *R*: $h \in$ *carrier R*
 and *diff-morph* $\in$ *carrier R*
 and *hh*: $h \otimes_2 h = \mathbf{0}_2$
 and *hdh*: $h \otimes_2$ *diff-morph* $\otimes_2 h = h$
 and *p*: $p = (h \otimes_2$ *diff-morph*$) \oplus_2 ($*diff-morph* $\otimes_2 h)$

 shows ((| *src* = *D*, *trg* = *D*(| *carrier* := *Ker D D* (*morph p*) |),
 morph = *morph* $(\mathbf{1}_2 \ominus_2 p)$, *src-comm-gr* = *D*, *trg-comm-gr* = *D* |))
 $\otimes_2 h$
 = (| *src* = *D*, *trg* = *D* (| *carrier* := *Ker D D* (*morph p*) |),
 morph = *morph* $\mathbf{0}_2$, *src-comm-gr* = *D*, *trg-comm-gr* = *D* |)

While working with elements of the ring R, i.e., tuples representing endomorphisms of D, expressions are manageable; homomorphisms, being represented with every field of the record, produce a less comprehensible notation.

The first of the premises states the conversion of the differential of D into an element of the ring of tuples R. The rest of the premises are directly extracted from the statement of Lemma 2.2.14, as well as the property in the goal.

For instance, the representation of $0_{\mathrm{Hom}(\ker p, D)}$ as a tuple is, as can be observed in the Isabelle statement:

(| *src* = *D*(| *carrier* := *Ker D D* (*morph p*) |), *trg* = *D*, *morph* = *morph* $\mathbf{0}_2$,
 src-comm-gr = *D*, *trg-comm-gr* = *D* |)

The null object is an endomorphism from the differential group D, and therefore the fields *trg-comm-gr* and *src-comm-gr* have D as value. We also must consider that the homomorphism we are representing is an inclusion from the differential subgroup $\ker p$ into the differential group D. Thus, the *src* field has value $\ker p$, and the *trg* field has value D, which is the codomain of the homomorphism. Finally, the homomorphism we are representing can be seen as the null object, which in Isabelle notation is the *morph* field of the endomorphism $\mathbf{0}_2$ (i.e., $\lambda x.$ *(if $x \in$ carrier D then zero D else zero D)*).

The method applied to give a proof of the property is as follows: first, computations are carried out with elements of the ring R of endomorphisms, taking advantage as far as possible of the Isabelle automation for such an algebraic structure. This equation must be expressed, consequently, in terms of elements of R; a combination similar to the one on the goal is required, but stated only with elements of the ring R. Then, by applying Lemma 3.7.1 in a suitable way, we convert the combination inside of the ring R into the linear combination appearing in the goal of the property we are proving.

The Isabelle proof itself, in practice, is not so clear in terms of readability due to some technical considerations. For instance, the proof of this property, required above 130 lines of Isar code. One of the reasons is that every time a conversion is needed from an element (or operator) of the ring R into the value this term has been instantiated with, a proof step is needed. For instance

from *prems* **have** *mult-def*: *op* $\otimes_2$ = *op* $\circ$
  by (*unfold group-mrp-set-ring-axioms-def group-mrp-set-def*) (*cases R*, *simp*)

is used to obtain that the concrete value with which op $\otimes_2$ is instantiated is op $\circ$ (an abbreviation of *group-mrp-comp*). Surprisingly, this difference between the syntactic and semantic meaning serves to illustrate the different reasoning levels we pretended to achieve. On a first level, we find a generic ring, which elements and operations have no further definition; on a second level, when the elements inside of this ring are explicitly interpreted, they become tuples representing the endomorphisms (and they cannot be used like elements of the ring). This situation, although clarifying how interpretation really works (i.e.providing *different* levels of abstraction requiring explicit processes to translate from one into the other), has as a consequence that numerous interpretations (and the opposite process) slow down proof performance, impoverish readability and increase proof size.

In a similar pattern, the rest of the properties can be expressed and proved, although with some limitations. The first property, stating that $(\mathrm{id}_{\mathrm{End}(D)} - p)\, \mathrm{inc}_{\ker p} = \mathrm{id}_{\ker p}$, would be expressed as follows:

lemma (**in** *group-mrp-set-ring*) *first-condition*:
 assumes *diff-morph*: *diff-morph* = (| *src* = *D*, *trg* = *D*, *morph* = *differ*,
          src-comm-gr = *D*, *trg-comm-gr* = *D* |)
  and *R*: *h* ∈ *carrier R*
  and *diff-morph* ∈ *carrier R*
  and *hh*: *h* $\otimes_2$ *h* = $\mathbf{0}_2$
  and *hdh*: *h* $\otimes_2$ *diff-morph* $\otimes_2$ *h* = *h*
  and *p*: *p* = (*h* $\otimes_2$ *diff-morph*) $\oplus_2$ (*diff-morph* $\otimes_2$ *h*)

shows $(\!|$ *src* $=$ *D*, *trg* $=$ *D*$(\!|$ *carrier* $:=$ *Ker D D* (*morph p*) $|\!)$,
morph $=$ *morph* ($\mathbf{1}_2 \ominus_2$ *p*) , *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$
$\otimes_2$
$(\!|$ *src* $=$ *D*$(\!|$ *carrier* $:=$ *Ker D D* (*morph p*) $|\!)$, *trg* $=$ *D*,
morph $=$ *morph* $\mathbf{1}_2$, *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$
$=$
$(\!|$ *src* $=$ *D* $(\!|$ *carrier* $:=$ *Ker D D* (*morph p*) $|\!)$,
trg $=$ *D* $(\!|$ *carrier* $:=$ *Ker D D* (*morph p*) $|\!)$,
morph $=$ *morph* ($\mathbf{1}_2 \ominus_2$ *p*), *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$

The proof is implemented in Isabelle following the same strategy used for the third condition previously shown. We carry on computations in the ring R, taking advantage of the automatized tactics; we start from the following composition (where each term belongs to the ring R of endomorphisms):

have *equality-in-ring*:
($(\!|$ *src* $=$ *D*, *trg* $=$ *D*, *morph* $=$ *morph* ($\mathbf{1}_2 \ominus_2$ *p*) , *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$
$\circ$ $(\!|$ *src* $=$ *D*, *trg* $=$ *D*, *morph* $=$ *morph* $\mathbf{1}_2$, *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$
$=$ $(\!|$ *src* $=$ *D*, *trg* $=$ *D*, *morph* $=$ *morph* ($\mathbf{1}_2 \ominus_2$ *p*), *src-comm-gr* $=$ *D*, *trg-comm-gr* $=$ *D* $|\!)$)

Then, with this part already proved, the implementation in Isabelle of Lemma 3.7.1 allows us to use the previous equality to prove a similar one where the sources and the targets of the endomorphisms match with the ones in the lemma's goal.

The implementation of Lemma 3.7.1 proved in *morph-comp-src-trg* in this case is the one modifying the source and the target of the endomorphism in the right hand side, and its implementation in Isabelle, together with its proof, is the following:

lemma *morph-comp-src-trg*:
assumes *A1*: *group-mrp2* $(\!|$ *src* $=$ *A*, *trg* $=$ *B*, *morph* $=$ *f*,
src-comm-gr $=$ *G1*, *trg-comm-gr* $=$ *G2* $|\!)$
and *A2*: *group-mrp2* $(\!|$ *src* $=$ *B*, *trg* $=$ *C*, *morph* $=$ *g*,
src-comm-gr $=$ *G2*, *trg-comm-gr* $=$ *G3* $|\!)$
and *A3*: (*carrier A'*) $\subseteq$ *carrier A*
and *A4*: *image g* (*carrier B*) $\subseteq$ *carrier C'*
and *A5*: (*group-mrp-comp* $(\!|$ *src* $=$ *B*, *trg* $=$ *C*, *morph* $=$ *g*,
src-comm-gr $=$ *G2*, *trg-comm-gr* $=$ *G3* $|\!)$
$(\!|$ *src* $=$ *A*, *trg* $=$ *B*, *morph* $=$ *f*,
src-comm-gr $=$ *G1*, *trg-comm-gr* $=$ *G2* $|\!)$)
$=$
$(\!|$ *src* $=$ *A*, *trg* $=$ *C*, *morph* $=$ *h*,
src-comm-gr $=$ *G1*, *trg-comm-gr* $=$ *G3* $|\!)$

```
  shows (group-mrp-comp    ⦇ src = B, trg = C', morph = g,
                             src-comm-gr = G2, trg-comm-gr = G3 ⦈
                            ⦇ src = A', trg = B, morph = f,
                             src-comm-gr = G1, trg-comm-gr = G2 ⦈)
                           =
                            ⦇ src = A', trg = C', morph = h,
                             src-comm-gr = G1, trg-comm-gr = G3 ⦈
proof (unfold group-mrp-comp-def, simp)
  from A4 show g o f = h by (unfold group-mrp-comp-def, simp)
qed
```

As it can be observed, the proof in Isabelle for this lemma is almost direct.

Both parts of the implementation of the proof in Isabelle of the first property took us above 100 lines; the first part required less effort, since the degree of automation is higher, and the computation inside of the ring R had a high level of automation. The application of Lemma 3.7.1 required a more specific work; the conditions needed made necessary even working with concrete elements of the homomorphisms' domain and codomain.

The second condition is a bit more complicated. The main reason is that more homomorphisms are involved in its statement: $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(D)} - p) + d_D h + h d_D = \mathrm{id}_{\mathrm{End}(D)}$. Despite of this difficulty, the complete proof can be implemented in Isabelle with a similar size to the previous ones. The Isabelle statement is expressed as follows:

```
lemma (in group-mrp-set-ring) second-condition:
 assumes diff-morph: diff-morph =    ⦇ src = D, trg = D, morph = differ,
                                       src-comm-gr = D, trg-comm-gr = D ⦈
  and R: h ∈ carrier R
  and diff-morph ∈ carrier R
  and hh: h ⊗2 h = 02
  and hdh: h ⊗2 diff-morph ⊗2 h = h
  and p: p = (h ⊗2 diff-morph) ⊕2 (diff-morph ⊗2  h)
  shows    ((⦇ src = (D ⦇ carrier := Ker D D (morph p)⦈), trg = D,
               morph = morph 12, src-comm-gr = D, trg-comm-gr = D ⦈
            ⊗2
             ⦇ src = D, trg = (D ⦇ carrier := Ker D D (morph p)⦈),
               morph = morph (12 ⊖2 p), src-comm-gr = D, trg-comm-gr = D ⦈)
            ⊕2 p = 12
```

The proof is done following a similar pattern as in the previous ones. Using automatized tactics, simplifications inside of the ring of endomorphisms R are carried out; we first operate on the composition of the endomorphisms, and then also on the addition, obtaining the unit endomorphism on the left hand side of the equation. In a second step, introducing a particular version of Lemma 3.7.1, it is proved that from the equality between endomorphisms, the equality between the homomorphisms holds.

The number of Isabelle lines of code was around 130, similar to previous properties, despite of the presence of a greater number of endomorphisms in the statement of the property. The size of the proofs could be decreased just by extracting some of the repetitive steps we have already highlighted, related to the interpretation (and the reverse process) of the ring elements, which are common to all proofs. A way to greatly improve the proof performance would consist in internally mechanize these steps, although this goes beyond our aim.

The fourth property that has to be proved following the definition of reduction was $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p,D)}$. In this property, we encountered some expected problems to give a complete implementation of the proof. Instead of proving the given statement, the property $h\,\mathrm{inc}_{\ker p} = hp\big|_{\ker p}$ had to be proved in Isabelle. The reason is that, when we are using the tuples, without modifying the meaning of the equality in Isabelle/HOL (which corresponds to the equality in the meta logic of the system), identities such as $p\big|_{\ker p} = 0_{\mathrm{Hom}(\ker p,D)}$ cannot be proved. The statement in Isabelle of this identity would be as follows:

(| *src* = (*D* (| *carrier* := *Ker D D* (*morph p*) |)), *trg* = *D*, *morph* = *morph p*,
src-comm-gr = *D*, *trg-comm-gr* = *D* |) =
(| *src* = (*D* (| *carrier* := *Ker D D* (*morph p*) |)), *trg* = *D*, *morph* = *morph* $\mathbf{0}_2$,
src-comm-gr = *D*, *trg-comm-gr* = *D* |)

Two tuples will be equal whenever the five fields of their records are equal. The equality in the algebraic structures *src, trg, src-comm-gr* and *trg-comm-gr* can be proved. When it is applied to the field *morph*, two functions will be equal when they are equal for all the elements of their source type. Taking into account that every endomorphism or homomorphism in this setting is represented through a completion function over the differential group D, and that $\ker p$ is a differential subgroup of it, the homomorphisms $p\big|_{\ker p}$ and $0_{\mathrm{Hom}(\ker p,D)}$ in this setting are not equal (they produce the same value when applied to the elements of the carrier set of $\ker p$, but not for every element of the carrier set of D). This can be understood as a limitation of representing all the endomorphisms and homomorphisms of the setting like endomorphisms of D, and then some rules allowing us to change their domain and codomain. On the other hand, we considered only the endomorphisms of D as necessary in order to keep the framework simple.

Thus, the statement of the fourth property that we have implemented in Isabelle is slightly different, establishing $hp\big|_{\ker p} = h\,\mathrm{inc}_{\ker p}$, instead of the original statement and its proof is as follows:

lemma (**in** *group-mrp-set-ring*) *fourth-condition*:
  assumes *diff-morph*: *diff-morph* = (| *src* = *D*, *trg* = *D*, *morph* = *differ*,
    src-comm-gr = *D*, *trg-comm-gr* = *D* |)
  and *R*: *h* ∈ *carrier R*
  and *diff-morph* ∈ *carrier R*
  and *hh*: *h* $\otimes_2$ *h* = $\mathbf{0}_2$
  and *hdh*: *h* $\otimes_2$ *diff-morph* $\otimes_2$ *h* = *h*
  and *p*: *p* = (*h* $\otimes_2$ *diff-morph*) $\oplus_2$ (*diff-morph* $\otimes_2$ *h*)

shows $h \otimes_2$
$(\!|$ *src* $=$ $(D$ $(\!|$ *carrier* $:=$ *Ker D D* $($*morph p*$)$ $|\!))$, *trg* $= D$,
morph $=$ *morph p*, *src-comm-gr* $= D$, *trg-comm-gr* $= D$ $|\!)$
$= h \otimes_2$
$(\!|$ *src* $=$ $(D$ $(\!|$ *carrier* $:=$ *Ker D D* $($*morph p*$)$ $|\!))$, *trg* $= D$,
morph $=$ *morph* $\mathbf{1}_2$, *src-comm-gr* $= D$, *trg-comm-gr* $= D$ $|\!)$

This proof required an intensive use of what we have defined as *equational reasoning* using the ring properties and the premises (this can be also observed in its mathematical proof in Lemma 2.2.14). The endomorphism h had to be successively converted, following the premises, into hdh, and then into $hdh + hhd$, which applying distributivity was $h(dh + hd)$ and thanks to the premises equal to hp. These computations in the set theoretic approach would have required a much greater effort. Thanks to the ring properties, all these steps were made almost automatically, just paying some special attention to the associativity laws in the Isabelle library.

The proof of the previous property required again around 130 lines of Isabelle code, and the methodology was similar to the previous properties. First, in the ring R of endomorphisms, we proved that $h\,\mathrm{id}_{\mathrm{End}(D)} = h$, and also that $hp = h$; then, by introducing Lemma 3.7.1, we obtained the same equalities, but for the homomorphisms in the statement of the property, namely, $h_{\ker p} = h\,\mathrm{inc}_{\ker p}$ and $h_{\ker p} = hp_{\ker p}$. Finally, by joining both equalities, the lemma was proved.

With this property, a complete implementation of the properties of reduction is finished. A complete implementation of the proof of Lemma 2.2.14 has been produced in Isabelle.

3.7.5 Discussion

The goal we pursued when this framework was proposed consisted in, first, avoiding the difficulties highlighted in the set theoretic approach for reasoning with homomorphisms (implemented through functions), and, at the same time, not to loose information about the homomorphisms and algebraic structures, as happened in the symbolic approach. The solution proposed here, among the ones already commented on is aimed to fit as far as possible with the previous works available in Isabelle. We did not want to modify the existing equality in Isabelle, since the consequences were unpredictable. Nevertheless, this remains as an open and interesting problem. We neither chose to define equivalence classes. As far as we know, equivalence classes have not been used yet in Isabelle for dealing with functions. The possibilities of implementation could be also studied.

Following the idea of the *extensional* functions defined in Isabelle, and due to their problems with the usual composition (as has been shown, extensional functions are not closed under composition), and the lack of a clear mathematical interpretation, we defined *completion* functions, functions which out of their domain of definition are assigned a distinguished element of their codomain. This idea has been already suggested and its consequences explored (see, for instance, [Harrison, 1996]). The intersection of these special functions with the Isabelle homomorphisms provided us with a set of functions over which we were able to define a ring of endomorphisms. The additive and multiplicative operators for this structure were the usual ones.

Once a ring was built, the automation provided by the symbolic approach was also recovered. In addition to this, information had not been lost related to the mathematical objects and structures involved in the problem. Thanks to these enhancements to the available Isabelle tools, endomorphisms can be dealt with like in usual mathematical proofs, especially when dealing with combinations of them.

In a second stage, and taking a closer look to Lemma 2.2.14, it can be observed that the two algebraic structures appearing on the statement have a special relationship. One of the differential groups is a differential subgroup of the other one. From an algebraic point of view, it can be seen as a closed subset that also satisfies some properties. The objects we are interested in now are the homomorphisms and endomorphisms between these two algebraic structures, which are the ones appearing in the proof of the lemma. Two possible representations can be chosen for this environment. A first one would consist in representing the homomorphisms and endomorphisms as belonging to different algebraic structures. Nevertheless, this situation does still have a difficulty: which are the operators allowing to combine endomorphisms and homomorphisms between the different algebraic structures? The operator we are looking for, in this concrete setting, is composition of functions, but a new question emerges: which one is the algebraic structure this operator belongs to, when it operates over endomorphisms and homomorphisms in the same expression?

This algebraic structure has been defined in the literature as ringoid (see Definition 1.1.24), although it is not very common. Its implementation in Isabelle could be given; it is not so clear how and to what extent reasoning in this structure could be automated.

A simplified implementation of the structure, with a fixed number of components will be presented in Section 3.8. In the morphism based approach, as has been referred, we took advantage of the relationship between the differential group and its differential subgroup, which allowed us to decrease the number of tools we had to implement. Then, representing endomorphisms through tuples and making an adequate use of Lemma 3.7.1, a satisfactory degree of functionality was obtained. A complete implementation of the proof of Lemma 2.2.14 was obtained.

The following facts can be highlighted from this proposal:

- A higher degree of functionality has been obtained for working with homomorphisms and endomorphisms. During the development of the framework we had to prove that homomorphisms between abelian groups form an abelian group with the binary operations of the underlying groups, and endomorphisms a monoid with the usual composition. Up to our knowledge, these proofs had never been done before in Isabelle. They increase the degree of automation, due to the existing libraries for algebraic structures in Isabelle.

- Some special lemmas, whose proofs in Isabelle were almost direct, helped us to deal with endomorphisms, and at the same time, with homomorphisms and endomorphisms of substructures.

- The degree of mechanization obtained in the proofs and their size was not as satisfactory as expected. One of the reasons was the necessity of making explicit the translations between the objects and operators of the algebraic structures, and the values they were interpreted with (endomorphisms and operators between them, in our case). A solution to this problem would be desirable, since these proof steps go against readability of proofs. A possible way out of this situation would be to introduce equalities between the abstract values of the locales fixed objects and their concrete meaning when defining the locale. Nevertheless, simplification processes should be careful with these "interpretation" steps, as far as they could produce infinite loops if applied automatically.

- The tools provided in this framework can be easily transferred to different problems in Group Theory. The representation of endomorphisms as a ring can result useful for both the lemmas we are facing and also other mathematical developments. We also have obtained the representation of homomorphisms between abelian groups like an abelian group, which also can be reused. The representation of endomorphisms and homomorphisms as tuples, where information about the source and the target is stored, has been proposed in some Category Theory approaches pretended in Isabelle (see for instance [Glimming, 2001, O'Keefe, 2004], where tuples were used to represent well defined arrows). Finally, the possibility of using equalities between endomorphisms to prove another equalities between endomorphisms and homomorphisms, modifying the domain and the codomain of endomorphisms through Lemma 3.7.1 can be applied, at least, to every environment where the objects appearing on the lemmas are homomorphisms and endomorphisms between an algebraic structure and as many substructures of it as needed.

3.8 The interpreting approach

A final method based on the interpretation of locales (see [Ballarin, 2004]), a tool recently implemented in Isabelle that should be helpful for the kind of theorems we are considering, is also explored. Interpretation of locales is a feature of the Isabelle release Isabelle2004. It allows us to define objects that fulfill the specification of a *locale* during the proofs.

Using this facility, we planned to define *locales* as close as possible to the mathematical context we have described, instead of proposing new tools, as in the previous approach (for instance, Lemma 3.7.1), that showed to be helpful to avoid keeping record of all the mathematical entities involved. In fact, some limitations were encountered while developing this environment. The first one is that the interpretation of locales was an ongoing project when we faced the problem. Therefore, we tried to make a simulation with the then existing tools. In addition to this, a precise description of the mathematical framework introduced in Lemma 2.2.14 requires algebraic structures not implemented in Isabelle. Once again, the precise description of the mathematical environment was not reachable.

Despite of these inconveniences, the approach was helpful, and showed to be more successful to complete the implementation of the proofs than the previous ones. Moreover, it should be easy to transfer it to other problems, even out of our interest area.

For testing these ideas, we again focused our attention on Lemma 2.2.14, which would allow us to compare the results obtained with the implementation of the proof proposed in the morphism based approach, in Section 3.7.4. We will skip in this approach the enumeration of the algebraic structures present in Lemma 2.2.14, since they have been already related in Section 3.7.1. In Section 3.8.1 we give a brief description of the representation of the mathematical structures involved in our problem; their implementation is based on code already introduced, and therefore details are omitted. Then in Section 3.8.2 we comment on the ideas that allow us to define a ring from a set of endomorphisms, and an abelian group from a set of homomorphisms, permitting us to use interpretation of locales to define an adequate environment. In Section 3.8.3 a complete implementation of the proof of Lemma 2.2.14 will be presented, improving the one already given in Section 3.7.4, mainly in terms of readability and code optimization. Finally, the comparison with the previous approaches, specially with the morphism based one, will be done in Section 3.8.4.

3.8.1 Representation of the algebraic structures

Two main issues have to be addressed in this section. First, which algebraic structures are to be considered in Isabelle when we implement our proofs and second, which implementation for these structures is chosen.

The second issue is solved as in previous approaches. Algebraic structures are represented through extensible records, which have shown to be appropriate and functional enough to fulfill our requirements.

On the contrary, with respect to the algebraic structures that are introduced in the framework, we take advantage of the facilities given by the interpretation of locales to create a *locale* object fitting to our problem, which can be interpreted with the algebraic structures present in the problem. In the *locale* we are defining in order to give a proof of Lemma 2.2.14, the following algebraic structures will be explicitly used:

- The differential group (D, d_D) (as in the morphism based approach).
- The differential subgroup $(\ker p, d_D)$, denoted in Isabelle by $Ker\text{-}p$ (as in the morphism based approach).
- A ring R, which will be interpreted with $\mathrm{End}((D, d_D))$ (as in the morphism based

approach).

- A ring R', interpreted with $\mathrm{End}((\ker p, d_D))$ (introduced in this approach).
- A commutative group A interpreted with $\mathrm{Hom}((D, d_D), (\ker p, d_D))$ (introduced in this approach).
- A commutative group A' interpreted with $\mathrm{Hom}((\ker p, d_D), (D, d_D))$ (introduced in this approach).

In comparison with the previous approaches, the amount of algebraic structures implemented in Isabelle has been increased. Here it is where the locales facilities for importing the available knowledge about the fixed structures and for referring implicitly to algebraic structures through the subindexes will be of great help. Actually, it can be say that *locales* do not augment the Isabelle reasoning power, although they ease the task of managing large proof contexts.

Now, in Section 3.8.2 a description of the implementation chosen for homomorphisms is given. It is mainly based on the ideas given in Section 3.7.3; the main difference is that now we do not store the source and the target algebraic structures of the homomorphisms, simplifying the Isabelle notation.

3.8.2 Homomorphisms between algebraic structures

Two main features determine the implementation of the homomorphisms we are seeking in this approach:

1. Homomorphisms, together with the necessary operations, have to be shown to form a ring or a commutative group (otherwise, interpretation would not be possible). Therefore, the ideas proposed in the morphism based approach in Section 3.7.3 allowing us to form a ring of endomorphisms are recovered here.
2. Thanks to the interpretation of locales, we have the possibility to automatize proofs in this setting, not only with the endomorphisms of the differential group (D, d_D), as in the morphism based approach, but also with the rest of homomorphisms and endomorphisms present in Lemma 2.2.14. Therefore, the implementation of homomorphisms through tuples proposed in the morphism based approach is no longer necessary, and only the functional information is relevant.

The proposed definition for homomorphisms, therefore, is similar to the functional part of the tuples introduced in the morphism based approach, which we know that satisfy both conditions:

```
constdefs
  hom-completion :: [('a, 'c) monoid-scheme, ('b, 'd) monoid-scheme] => ('a => 'b)set
```

$hom\text{-}completion\ G\ G' ==\ \{h.\ h \in completion\text{-}fun2\ G\ G'\ \&\ h \in hom\ G\ G'\}$

This Isabelle definition was already proposed in Section 3.7.3. The type assigned to homomorphisms is a functional one, between the types of the source and the target structures. The definition can be divided into two parts. The first part, already proposed in the set theoretic approach in Section 3.6.3, stating that every homomorphism must assign to each element of the source set one element of the target set, and also that every homomorphism must be coherent with the binary operations of the source and target algebraic structures. The second part, added in Section 3.7.3 is used again: every element out of the source set is mapped to the unity of the target algebraic structure. This definition allowed us to prove that endomorphisms and homomorphisms, with appropriate operations, define certain algebraic structures. For instance, the following structure, with the usual composition of functions, forms a monoid:

$(|\ carrier = hom\text{-}completion\ G\ G,\ mult = op\ o,$
$one = (\lambda x.\ if\ x \in carrier\ G\ then\ id\ x\ else\ one\ G)|)$

We also proved that an abelian group can be defined making use of the binary operation of the underlying group G' as follows. In the case of endomorphisms, it can be considered $G' = G$. Otherwise, the following algebraic structure can be seen as the set of homomorphisms between two abelian groups G and G':

$(|\ carrier = hom\text{-}completion\ G\ G',$
$add = \lambda f.\ \lambda g.\ (\lambda x.\ if\ x \in carrier\ G\ then\ f\ x \oplus_2\ g\ x\ else\ \mathbf{0}_2),$
$zero = (\lambda x.\ if\ x \in carrier\ G\ then\ zero\ G'\ else\ zero\ G')|)$

By proving also the distributivity of the multiplicative operation with respect to the additive one, we can prove that the set of endomorphisms of a given commutative group G defines a ring with the previous operations.

These facts are the ones allowing us to interpret the *locale* we are trying to define with the given structures based on homomorphisms. Therefore, once these facts have been proved, the next step will be to define the concrete *locale*, as a final step before developing the proofs.

3.8.3 Lemmas proved

The definition of a *locale* containing the mathematical entities present in Lemma 2.2.14 can be now done. In the following definition, the *locale* object is defined, and at the same time, its variables are interpreted with the objects that reproduce the mathematical setting in the lemma we are facing:

locale $hom\text{-}completion\text{-}ringoid = abelian\text{-}group\ D + abelian\text{-}group\ Ker\text{-}p + ring\ R$
$+\ ring\ R' + abelian\text{-}group\ A + abelian\text{-}group\ A' + var\ p\ +$

```
  assumes R = (| carrier = hom-completion D D,
                 mult    = op o,
                 one     = (λx. if x ∈ carrier D then id x else zero D),
                 zero    = (λx. if x ∈ carrier D then zero D else zero D),
                 add     = λf. λg. (λx. if x∈ carrier D
                                    then f x ⊕ g x else zero D)|)
  and R' = (| carrier = hom-completion Ker-p Ker-p,
              mult    = op o,
              one     = (λx. if x ∈ carrier Ker-p then id x else zero Ker-p),
              zero    = (λx. if x ∈ carrier Ker-p then zero Ker-p else zero Ker-p),
              add     = λf. λg. (λx.if x ∈ carrier Ker-p
                                 then f x ⊕ g x else zero Ker-p)|)
  and A = (| carrier = hom-completion D Ker-p,
             add     = λf. λg. (λx. if x ∈ carrier D then f x ⊕ g x else zero Ker-p),
             zero    = (λx. if x ∈ carrier D then zero Ker-p else zero Ker-p) |)
  and A' = (| carrier = hom-completion Ker-p D,
              add     = λf. λg. (λx. if x ∈ carrier Ker-p then f x ⊕ g x else zero D),
              zero    = (λx. if x ∈ carrier Ker-p then zero D else zero D) |)
and subgroup2 Ker-p D
and (carrier Ker-p) = {x. p x = 0₂}
```

The name of the *locale*, *hom-completion-ringoid*, is due to the resemblance of the context created with the ringoid structure. The *locale* fixes two abelian groups D and $Ker\text{-}p$, and then a ring R is also fixed and interpreted with the endomorphisms of D, a different ring R' is fixed and interpreted with the endomorphisms of $Ker\text{-}p$, an abelian group A with the homomorphisms from G into $Ker\text{-}p$, and an abelian group A' with the homomorphisms from $Ker\text{-}p$ into D. In addition to this, $Ker\text{-}p$ is defined to be a subgroup of D, and with carrier set the elements which image through p is the identity of D. The previous lemmas proving that for any abelian groups G, G', $\mathrm{End}(G)$ form a ring and $\mathrm{Hom}(G, G')$ an abelian group with the proposed operations, ensure the correctness of the definition of the given *locale*. With a working version of the interpreting tool, these lemmas should be applied in order to prove the correctness of the *locale* definition.

Once we have defined a setting, the properties in Lemma 2.2.14 have to be proved. The proof methodology exposed in the morphism based approach can be applied also here. As far as the endomorphisms and homomorphisms can be understood as belonging to certain algebraic structures, computations can be carried out in these structures taking advantage of the automation available. For instance, this has been applied to the third property, which statement and proof in Isabelle, when we are in the *locale* *hom-completion-ringoid* is as follows:

```
lemma (in hom-completion-ringoid) reduction-property-three:
  assumes p ∈ carrier R and h ∈ carrier R and  p ⊕₃ h = h
shows (one R ⊖₃ p) ⊕₃ h = zero A
proof −
  from prems have one: ((one R ⊖₃ p) ⊕₃ h = zero A) = ((one R ⊖₃ p) ⊕₃ h = zero R)
  proof –
    from prems have zero A = zero R
      by (unfold hom-complection-ringoid-axioms-def subgroup2-def)(cases A, cases R, cases
Ker-p, auto)
    then show ?thesis by simp
  qed
  from prems have two: (one R ⊖3 p) ⊕3 h = zero R
    by algebra
  from one and two show ?thesis by simp
qed
```

The proof corresponds to the property $(\mathrm{id}_{\mathrm{End}(D)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\ker p}$. In the proof, *zero A* is the zero field in the abelian group A fixed in the *locale*, i.e.:

zero = (λx. if x ∈ carrier G then zero Ker-p else zero Ker-p)

It can be proved to be equal to *zero R*. Once this has been proved, all the objects in the proof belong to the ring R, and the tactic *algebra* is able to solve the goal.

A fact that can be also pointed out is the difficulties found in Isabelle to simplify atomic parts of combinations. For instance, in the expression we are working with now, despite being able to prove the following equality in a single proof step,

zero A = zero R

the equality between the following combinations requires a stepwise proof:

((one R ⊖3 p) ⊗3 h = one A) = ((one R ⊖3 p) ⊗3 h = zero R)

First, the previous equality between atomic terms must be supplied, and then the equality between the expressions can be derived. Simplification tactics, without being explicitly instantiated, are not enough to deal with the equality between both expressions.

The rest of the properties have more elaborate proofs, at least in number of lines of Isabelle code, since the automation tactics cannot be applied so directly. The statement of the first property, which was $(\mathrm{id}_{\mathrm{End}(D)} - p)\,\mathrm{inc}_{\ker p} = \mathrm{id}_{\mathrm{End}(\ker p)}$, with the supplied tools, remains as follows:

```
lemma (in hom-completion-ringoid) reduction-property-one:
  [| p ∈ carrier R |] ==> (one R ⊖3 p) ∘
  (λx. if x ∈ carrier Ker-p then id x else zero Ker-p ) = one R'
```

Two reasons can be suggested that prevented us of seeking a fully automatic proof in Isabelle of this lemma, and therefore, augmented the size of its proof. First, the minus operator does not count with a lot of automated tactics, and requires a very careful treatment. Secondly, the equality between both expressions cannot be easily transferred to one of the algebraic structures involved in the proof (rings R and R', or abelian groups A and A'). As far as no automation has been given for the implicit ringoid present in the mathematical environment of the proof, most of the steps had to be done making use of very basic Isabelle tactics (relative to the properties and definition of $\circ$ and extensionality principles). The whole proof required above 90 lines of code.

The same situation appears in the rest of the properties. The statement of the second property was $\mathrm{inc}_{\ker p}(\mathrm{id}_{\mathrm{End}(D)} - p) + d_D h + h d_D = \mathrm{id}_{\mathrm{End}(D)}$ and its implementation in this framework is:

lemma (**in** *hom-complection-ringoid*) *reduction-property-two*: $[| \ p \in \mathit{carrier}\ R;\ p \otimes_3 p = p \ |]$
 $\texttt{==>}\ ((\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ \mathit{Ker\text{-}p}\ \mathit{then}\ \mathit{id}\ x\ \mathit{else}\ \mathit{zero}\ \mathit{Ker\text{-}p}\ ) \circ (\mathit{one}\ R \ominus_3 p)) \oplus_3 p = \mathit{one}\ R$

The size of its proof is 40 lines. The sketch of the proof is similar to the one given in the morphism based approach for this lemma. We try to reduce the left hand side of the expression in the goal, first by simplifying the composition, and then by eliminating the endomorphism p, until we obtain the expression on the right hand.

The same process is applied also to the fourth property, stating that $h\,\mathrm{inc}_{\ker p} = 0_{\mathrm{Hom}(\ker p, D)}$, which Isabelle statement is:

lemma (**in** *hom-completion-ringoid*) *reduction-property-four*:
 $[| \ p \in \mathit{carrier}\ R;\ h \in \mathit{carrier}\ R;\ h \otimes_3 p = h \ |]$
 $\texttt{==>}\ h \circ (\lambda x.\ \mathit{if}\ x \in \mathit{carrier}\ \mathit{Ker\text{-}p}\ \mathit{then}\ \mathit{id}\ x\ \mathit{else}\ \mathit{zero}\ \mathit{Ker\text{-}p}\ ) = \mathit{one}\ A'$

Again the proof is based on very low level Isabelle principles, introducing extensional equality between functions to produce the proof. The prover is capable of discarding the non-meaningful cases, remaining only the case where $x \in \mathit{carrier}\ \mathit{Ker\text{-}p}$. The proof of this case requires less than 30 lines.

The fifth property, stating that the homotopy operator h is nilpotent, is among the premises. With the five properties gathered together, the proof of Lemma 2.2.14 is completed.

3.8.4 Discussion

Some remarks can be done dealing with this interpreting approach. In relation to the previous approaches, the following can be said:

- This approach is based on considering endomorphisms and homomorphisms as algebraic structures, instead of sets. We had already introduced this idea in the morphism based approach, where we had to develop the tools allowing to define both the representation for homomorphisms and also the operators allowing to define the algebraic structures (rings and commutative groups). On the contrary, in this approach, the environment built is less flexible than in the morphism based approach. There, we provided lemmas allowing us to modify the source and target of homomorphisms under some weak premises. Here, on the contrary, all the homomorphisms and endomorphisms are defined when the *locale* object is created, and afterwards, it is not possible to modify a single homomorphism; just computations between various homomorphisms can be carried out (taking advantage of the ringoid we have defined).

- The code obtained in our proofs has been substantially decreased. The number of code lines in relation to the morphism based approach has been now divided by 4 approximately; one first reason is that representation of homomorphisms is now simpler. As far as homomorphisms now are represented just by their functional part, the size of their codification is smaller. A second reason can be also given. In the morphism based approach, automation was only possible with the operations inside of the ring; otherwise, the proofs with records had to be managed carefully by hand. Here, the objects we are dealing with are mainly functions, and the previous knowledge in the Isabelle library related to basic operations with functions is directly applied; for instance, properties about the composition, extensional principles expressing the equality between functions, management of lambda expressions, are already introduced in the simplifier. As far as the combinations we deal with in these lemmas are not very complicated, goals can be almost fully simplified without help. Due to this reason, proofs are less elegant than in the morphism based approach; there, computations were carried out in a ring (based on equational reasoning or term rewriting) and then transferred to their original form. Now, expressions are considered in their original form (seen as combinations of objects belonging to different structures), and proofs are based on powerful tactics automatically discarding most of the possibilities.

- The abstraction process from the mathematical entities to the objects we have implemented is now a bit more obscure. In the morphism based approach, every homomorphism and endomorphism could be understood as an object of a ring, and the operations where identified with the ring operators, and the identification between mathematical entities and objects implemented in the theorem prover was clear. Now, in the interpreting approach, the algebraic structures and their operations in the theorem prover have a clear identification with the mathematical entities they represent. On the other hand, the operation allowing to operate homomorphisms of the different algebraic structures, i.e., $\circ$, does not belong to any algebraic structure. In the mathematical setting, the operation $\circ$ belongs to the ringoid which encloses the different algebraic structures. But this ringoid has been avoided in the translation to the theorem prover. This has a direct consequence. As far as the ringoid has not been implemented, the only information and properties about the $\circ$ operator are the ones available from the Isabelle libraries. On the other hand, in the morphism based approach, the $\circ$ operator was identified with the multiplicative operation inside of the ring of endomorphisms. It seems that when we try to work with large combinations of endomorphisms and homomorphisms, the automation obtained in the morphism based approach will be greater than in this approach.

The ideas exposed and the tools supplied to Isabelle were enough to complete the proof of the properties in Lemma 2.2.14. Nevertheless, some other ideas have not been fully explored, and should be further studied:

- Taking into account that interpretation of locales was not available when we tried these ideas, a new effort should be made to translate these proofs into the recently implemented tool, even if no substantial changes are expected from what we have already obtained. Interpretation of locales does not increase the reasoning power of the system, but it greatly improves the syntax of the proofs as well as their size; its possibilities go beyond the scope of our problem, and can be applied to every kind of environment.
- Ringoids appear in several abstract algebra problems. Some effort could be made to obtain an implementation of them in the theorem prover. Even if one is able to implement them in Isabelle, obtaining some degree of automation in this structure seems far from trivial.

Chapter 4

Extracting Computer Algebra Programs from Statements

4.1 Introduction

Kenzo is a Common Lisp program created by Sergeraert (see [Dousson et al., 1999] and also Sections 1.2.1, 1.2.2 of this memoir), for Computer Algebra computations in the field of Algebraic Topology. Its main characteristics are its handling of infinite spaces (by using functional programming), and that Kenzo has found results unreachable by any other means (see [Rubio and Sergeraert, 1997, Rubio and Sergeraert, 2002]).

In Chapter 3 we have presented various approaches to deal with objects of Algebraic Topology, such as differential groups and homomorphisms between them, that were useful to implement fragments of the proof of the BPL in Isabelle. Taking into account that our long term goal was to increase the reliability of the Kenzo system by giving certified versions of the crucial algorithms present on it, the next step of our research consists in obtaining programs from the proposed mechanized proofs. The obtained programs should be then compared with the original ones present in the Kenzo system and encoded in CLOS (the Common Lisp object oriented system) and proved to be equivalent, i.e., they should produce similar results for similar inputs; moreover, they would be certified to be correct, as far as we rely on the code extraction tool. The importance of this certification is clear, since some of the results obtained with Kenzo in Homological Algebra have not been obtained by any other means.

One of the reasons that we claimed for using Isabelle as our theorem prover was the code extraction tool that has been implemented on top of it (see [Berghofer, 2003a, Berghofer, 2003b, Berghofer, 2004]). This tool can be applied to a subset of Isabelle/HOL satisfying certain constraints.

From the different approaches presented in Chapter 3, the one fitting most exactly to our purposes was the morphism based approach (see Section 3.7), as far as it provided us with a high degree of expressiveness and also of automation. There, homomorphisms were implemented as extensible records encoding the real map, the potential source and target chain complexes, an also the real domain of definition and the image.

Despite of the success of this approach, our aim was to obtain a formalization (and proof mechanization) of the *theorems*, not of the *programs* appearing in Kenzo. Therefore, now we try to bridge the gap between (mechanized) theorems and programs using Berghofer's tool for extracting ML programs from Isabelle theories. We suspected that the proving efforts previously done could perhaps be unsuitable, due to the additional constraints on the constructive nature of the proofs (up to now, we have chosen a *classical* way of proving in Isabelle, trying to emulate the proofs-by-hand from Homological Algebra; see [Rubio, 2004]). Surprisingly enough, we observed that most of our already formalized theorems had *constructive statements* (in a sense that will be explained in Section 4.3), even if proofs are not necessarily expressed in a constructive manner. This simple observation allows to apply Berghofer's tool to some of our Isabelle theories, extracting ML programs equivalent to some (small) fragments of Kenzo. Even if preliminary (the programs extracted so forth are extremely simple, compared with Kenzo as a whole), these results invite to explore further this research line.

The chapter is organized as follows. Section 4.2 is devoted to introduce the case study chosen from Kenzo: the composition of two homomorphisms. A fragment of CLOS code will be shown where the composition is defined in the Kenzo system. In Section 4.3, we move to a well-known domain, namely elementary arithmetic, to work out a simple example related to Euclid's proof of the existence of infinitely many primes. The aim of this section is to introduce some key ideas, avoiding the complexities of Homological Algebra and Algebraic Topology; basics on formalization, automated theorem proving and program extraction are also introduced, including our notion of *constructive statement*. Then, this notion is applied to the elementary arithmetic example, showing how Berghofer's tool can be used to obtain an ML program (certified correct) computing a prime number bigger than its input. In Section 4.4, we recover our original framework, going back to our Homological Algebra proofs, applying the same techniques to obtain an ML program (certified correct by the Isabelle tool) to compose two homomorphisms.

4.2 The Kenzo program: some fragments

In [Rubio, 2005] a fragment of the implementation in Kenzo of the BPL is presented. Kenzo is a quite complex Common Lisp program, over 16000 lines of CLOS code with intensive use of functional programming techniques. In this section, we will focus our attention on the fragments of Kenzo code that define the composition of homomorphisms. Even in this case, some explanations are needed in order to understand the code. A piece of code of the Kenzo implementation of the BPL will be also shown. As far as we would like to achieve an implementation of the algorithm associated to the proof of this lemma with a code extraction tool, we considered interesting to introduce a fragment of its implementation in Kenzo to illustrate the complexity of our task.

In Kenzo every chain complex is *free*. Consequently, every graded group in Kenzo is defined from its set of *generators* and an *equality test* among them for *each degree*. Generators are used to form *combinations* (which are linear combinations of generators, with coefficients ranging over the integer numbers), which are the real elements of the graded group for each degree. In order to add two combinations, the equality test between generators has to be used. With this implementation in CLOS, when we define a homomorphism between two chain complexes, it is enough to provide the image of each generator, that can be identified with a linear combination of the generators of the target graded group. In order to extend the map to combinations on the source group, the equality test *in the target group* has to be used. This strategy, which is the most frequently used in Kenzo, is called, according to Sergeraert's terminology, *by generator*, and is denoted in the Kenzo system with the keyword `:gnrt`.

But there are cases where this method can be wasteful from the performance point of view: for instance, the identity or the null homomorphisms do not require any equality checking on the generators. A second strategy can be considered that Sergeraert called *by combination* (defined as `:cmbn` in Kenzo), that solves the previous performance problems. The strategy followed can be indicated explicitly in the composition, although it is not necessary. This explains the optional parameter (called `strt` for *strategy*) in the following Kenzo program

```
(DEFMETHOD CMPS ((mrph1 morphism) (mrph2 morphism) &optional strt)
   ;;; ... lines skipped
  (build-mrph :sorc sorc2 :trgt trgt1 :degr (+ degr1 degr2)
              :intr #'(lambda (cmbn)
                         (declare (type cmbn cmbn))
                           (the cmbn
                               (cmbn-? mrph1 (cmbn-? mrph2 cmbn))))
              :strt :cmbn :orgn `(2mrph-cmps ,mrph1 ,mrph2 ,strt))
   ;;; ... lines skipped
```

The program is defined to be a *method* since Kenzo is written in CLOS, which uses object oriented features, that allowed Sergeraert, by using the common inheritance mechanism, to give the same name to similar methods which compose two homomorphisms between coalgebras and another related algebraic structures. The lines skipped correspond to the cases in which one of the two homomorphisms has (or both have) a strategy by generator.

The fragment showed here corresponds to the case where at least one of the homomorphisms has as strategy *by combination*, and this is also the strategy in the result homomorphism: `:strt :cmbn`. The variables `sorc2`, `trgt1`, `degr1` and `degr2`, correspond, respectively, to the source of the second homomorphism `mrph2`, to the target of the first one, and to the degrees of the homomorphisms, which have been previously defined from the parameters in the lines skipped. The fragments `(declare (type cmbn cmbn))` and `the cmbn`, show that we are in a *typed* context. The keyword `:orgn` is used to store some information about the *origin* of the new homomorphism (that is to say, on the method and arguments constructing the new object), for software engineering purposes. Finally, the Kenzo function `cmbn-?` allows the programmer to invoke a homomorphism on a combination, and it is used here (twice) to accomplish the actual composition.

Avoiding the technicalities, the essence of the previous method in the particular case of chain complexes homomorphisms of degree zero, is equivalent to the following Common Lisp function:

```
(defun CMPS (g f)
  (build-mrph :sorc (sorc f) :trgt (trgt g)
              :intr #'(lambda (cmbn)
                          (cmbn-? g (cmbn-? f cmbn)))))
```

The process of formalization will be illustrated with this simplified version. As it has been previously explained, the proofs of the theorems in Section 2.2.2 can be carried out with ungraded structures. The piece of Common Lisp code shown above defines the composition of homomorphisms where the degree information is missed. As it could be expected, for defining `f` composed with `g`, the source is given by the source of `f`, the target by the target of `g`, and the functional part is defined like the composition of both functions.

The complexity of the algorithm implementing the BPL in Kenzo is much greater than the previous function. Here we have extracted a fragment of it, just in order to have a look on the difficulties that could be found before arriving to a complete version of a certified version of it extracted from Isabelle code automatically:

```
(DEFUN BPL-*-sigma (homotopy perturbation)
   (declare (type morphism homotopy perturbation))
   (the morphism
      (let ((cmpr (cmpr (sorc perturbation)))
            (h-delta (cmps homotopy perturbation)))
         (declare
            (type cmprf cmpr)
            (type morphism h-delta))
         (flet
            ((sigma-* (degr gnrt)
                (declare
                   (fixnum degr)
                   (type gnrt gnrt))
                (do ((rslt (zero-cmbn degr)
                           (2cmbn-add cmpr rslt iterated))
                     (iterated (term-cmbn degr 1 gnrt)
                               (cmbn-opps (cmbn-? h-delta
                                                  iterated))))
                    ((cmbn-zero-p iterated) rslt)
                   (declare (type cmbn rslt iterated)))))
            (build-mrph
               :sorc (sorc homotopy) :trgt (sorc homotopy)
               :degr 0 :intr #'sigma-* :strt :gnrt
               :orgn `(bpl-*-sigma ,homotopy ,perturbation))))))
```

Figure 4.1: BPL fragment in Kenzo

4.3 Elementary examples

As the previous Common Lisp fragments of Kenzo have shown, dealing with algorithms in Homological Algebra has an additional difficulty related to the complexity of the studied structures. In order to illustrate our ideas, the difficulty of the area will be avoided by moving to simpler problems in well-known settings, such as group theory or elementary arithmetic. In this section, the question of proving the correctness of a program is also studied.

The first example will be useful to present the code extraction tool which is being implemented on top of Isabelle. A basic lemma about monoids is introduced, whose proof in Isabelle is also given. From the Isabelle statement of this lemma, the code extraction tool will be capable of extracting an algorithm, certified correct and associated to the proved lemma.

Lemma 4.3.1. *Let G be a monoid and $*$ its binary operation. Let us define the operation $*_{rev} = \lambda x.\lambda y.y * x$, where x and y are elements of G. Then, the carrier set of G together* *with $*_{rev}$ and the unit of G form a monoid.*

In order to express the statement of this lemma in Isabelle, we first define a function in Isabelle called *rev* which, given a monoid G, defines a new monoid where the binary operation has been modified as proposed:

constdefs
  rev :: ('*a*, '*m*) *monoid-scheme* => '*a monoid*
  rev G == (| *carrier* = *carrier G*, *mult* = ($\lambda x\ y.$ *mult G y x*), *one* = *one G* |)

Then, the statement and proof in Isabelle of Lemma 4.3.1 can be expressed as follows:

lemma (**in** *monoid*)
  shows *monoid* (*rev G*)
  apply (*unfold rev-def*)
  apply (*rule monoidI*)
  apply (*auto simp*: *m-assoc*)
  done

The proof is done by unfolding the definition of the *rev* function, and then also the one of monoid, and introducing the associativity of the binary operation. Now, it can be observed that the statement of the lemma fully determines the monoid we are trying to define. There is no need to explore inside of the proof to know about how the monoid has been built. Thus, our approach to the code extraction process can take advantage of this particular circumstance to obtain code from the definition of the *rev* function. Thanks to the proved lemma, we know that this function, when applied to a monoid, will return also a monoid.

The Isabelle code extraction tool produces ML code; when applied to the Isabelle definition of function *rev*, the following ML definitions are automatically extracted:

```
fun one r = fst (snd (snd r));

fun mult r = fst (snd r);

fun carrier r = fst r;

fun Monoid_rev G = (carrier G, ((fn x => fn y => mult G y x), (one
G, Unity)));

val rev = Monoid_rev;
```

Functions `one`, `mult` and `carrier` are just projectors of the corresponding components of a given record. Whenever the components of the record satisfy the requirements of a monoid, the `Monoid_rev` function will build a new monoid with the binary operation modified in the sense we previously defined. The `Unity` component resembles the extensible part of records in Isabelle. As far as the Isabelle definition of monoids we have provided is generic, with information just about the arities of the operations and the axiomatic specification, the function extracted is also generic over lists of any type.

Now we face a more concrete example, where types are specified. We define in Isabelle (through a function) an algebraic structure whose carrier set will be the integers (objects with *int* type in Isabelle), with binary operation the addition (+) and with 0 as unit. Its Isabelle definition is as follows:

constdefs
  intg :: *int monoid*
  intg == (| *carrier* = *UNIV*, *mult* = ($\lambda x\ y.\ x + y$), *one* = *0* |)

The carrier set of the structure is the set of all integers, here represented as *UNIV*, whereas the operation is usual addition and the unit is 0. It is almost a trivial proof in Isabelle to show that this algebraic structure is a monoid.

Some considerations have to be done before extracting code from function *intg*, referred to the representation of sets in ML. In Isabelle, sets are internally represented as predicates over their type, and this is also the representation we would like to have in ML. Representation of sets is relevant because *UNIV* is an Isabelle set containing all the objects of *int* type. In order to achieve this representation, we must specify in our Isabelle code the following command:

types-code
  set ((- −> *bool*))

The previous command is a kind of translation for the Isabelle type constructor *set*. The unspecified type represented as - will be substituted by the type of the objects of a concrete set (*int* in this example).

Once we have specified the definition we have chosen for sets in Isabelle, code can be extracted from the definition of the *intg* function. The obtained code is as follows:

```
datatype unit = Unity;

val UNIV : (int -> bool) = ((fn x => true));

val intg : ((int -> bool) * ((int -> (int -> int)) * (int * unit)))
= (UNIV, ((fn x => fn y => (x + y)), (0, Unity)));
```

A new data type `unit` is defined with only one value, `Unity`. This resembles the extensible records from Isabelle. A function `UNIV` is declared and defined (based in the representation we chose for sets) in the form of a predicate. Finally, the `intg` function is also assigned a record type. In comparison with the previous example, now the ML types `int` and `bool` appear in the definition of the algebraic structure `intg`, as well as the ML operation `+` and the constant `0`.

The previous examples were quite straightforward. The definitions introduced were precise enough to be compatible with the code extraction tool. Now we present a different example, where we will have to modify standard definitions in order to obtain functions from which code can be extracted. This example deals with some well-known facts about prime numbers. We want to obtain a program which, taking any natural number as parameter, returns a prime number greater than this parameter. This program is related to the Euclid's theorem about the infinity of primes. Plenty of proofs have been given of this theorem, very different among them, with some of them closer to constructivism and some of them a bit more distant.

The following Common Lisp program computes the next prime to a given positive integer number $x > 1$.

```
(defun nextprime (x) ; x integer > 1
  (if (isprime (+ x 1))
      (+ x 1)
    (nextprime (+ x 1))))
```

Here, it is assumed the existence of a Common Lisp program `isprime` which determines if its input is a prime number. In addition, we assume that `isprime` is correct. This will be used in the following elementary result.

Theorem 4.3.2. *The program* `nextprime` *is correct with respect to the following specification:*
Input specification: x *integer*, $x > 1$.
Output specification: `(isprime (nextprime x))` *returns* true.

Proof. The proof is divided into two parts:

1. *Partial correctness.* If the program returns a value, `(nextprime x)` $= z$, with $z =$ `(+ y 1)` and `(isprime (+ y 1))` $=$ *true* (due to the semantics of `if`). Then, the output specification follows.

2. *Proof of termination.* By contradiction.
Let us assume that there exists x integer, $x > 1$, such that `(nextprime x)` does not terminate. This implies that for all y such that $y > x$, y is not a prime number. Let $P = \{p_1, \ldots, p_n\}$ be the set of primes smaller than x. By hypothesis, this is the set of *all* prime numbers. Let us consider $m = p_1 * \ldots * p_n + 1$. Then we have that p_i does not divide m, $\forall i = 1, \ldots, n$. We conclude, by applying Lemma 4.3.3 (see below), that m is prime. It is clear that $m > p_i$, $\forall i = 1, \ldots, n$, and thus $m \notin P$ and consequently is not a prime number. *Contradiction.*

■

Lemma 4.3.3. *Let m be an integer number, $m > 1$. Then, m is prime if and only if for every prime number $p < m$, p does not divide m.*

In the previous proof it can be perceived Euclid's argument showing that there are infinitely many primes. Nevertheless, there are several variations on Euclid's idea. This one has the property that the computational content is more hidden than in other variants, which are based on Lemma 4.3.4, instead of on Lemma 4.3.3. In those proofs, even if presented in a "reductio ad absurdum" manner, the computational content is quite explicit[1].

Lemma 4.3.4. *For each integer $m > 1$, there exists a prime number $p \leq m$ such that p divides m.*

Such a detailed proof of correctness has been provided since our aim is not only to give such a proof, but also a *certificate of correctness*. Even more specifically, we are looking for Isabelle scripts containing proofs of correctness for our programs. To this aim, it is necessary to link in some way the running code (or, at least, the source code) with some formalization of it. In a generic setting this task is far from trivial (the lack of the sought link has been illustrated graphically in the previous proof, where the symbols `x` and x have been used in an undistinguished manner). The following strategies, based on the use of code extraction tools, can be useful to couple programs and their formalization.

In order to build correctness certificates for programs, a first necessary task is to formalize the objects of study in a computer-aided mathematical tool. In the elementary example in arithmetic we are considering, both Mizar and Isabelle are known to be adequate (in fact, the arithmetic example has been extracted from [Wenzel and Wiedijk, 2002], where it is used to compare Mizar and Isabelle). However, once the objects of study (i.e., the proofs) have been formalized, new efforts are needed in order to link the formalization with a *real* program.

A well-known strategy is to establish the formalization in a (mechanized) *constructive* logic, and then extract a program from the proof. This is the point of view, for instance, when the Coq system is used for this task. Nevertheless, it is not the *only* way to get certificates for programs. Here we present a second strategy that we first introduced in [Aransay et al., 2005].

In particular, the Common Lisp program `nextprime` and the non-constructive proof of correctness given in Theorem 4.3.2, seem to be far from the first strategy (up to our knowledge, the program cannot be obtained from its proof of correctness using one of the mentioned tools). This is true from a strict-constructivist perspective,

[1]Thanks are due to W. Bosma, who explained in the Map e-list (see `http://www.disi.unige.it/map/`) that the same is not true in the proof presented above.

but it is not in Markov's *constructive recursive mathematics*[2]. In fact, the program `nextprime` is a paradigmatic example of a (correct) program generated by *Markov's principle* [Markov, 1971]. Markov's principle can be stated as $\neg\neg\exists x.A(x) \rightarrow \exists x.A(x)$, where A is primitive recursive, and is applied for showing the termination of computations by contradiction: if a computation cannot diverge then it must terminate (which is the argument we considered for ensuring the termination of function `nextprime`). These issues are also discussed in [Kopylov and Nogin, 2001], where Markov's principle is integrated with constructive type theory, and by Berghofer in [Berghofer, 2005], in the concrete context of his extraction tool from Isabelle scripts.

Taking into account this example, it seems that Davenport's observation in [Davenport, 1981] (or, even more explicitly, in [Davenport, 1989, page 140]), claiming that *Computer Algebra* correctness proofs can be done by not necessarily *strict-constructive* methods, is quite accurate (up to our knowledge, whether proofs in Computer Algebra can go further than Markov's constructivism is an open problem).

In fact, in the field of algorithmic homological algebra it has been observed that the theorems that we try to formalize have *constructive statements*. That is to say: a new object is defined (the composition of two homomorphisms, in our running example) and some properties of this object are asserted (namely, that it is a homomorphism). Then, code can be extracted from the definition or specification appearing in the statement. Thus, this approach seems to indicate that the underlying logic which is used to implement the proof is not forced to be constructive (since the program is extracted from definitions and not from proofs), in the vein of Davenport's observation. This strategy will be presented in the case of the composition of homomorphisms in Section 4.4. First, we reconsider the arithmetic example, to show how in this case definitions can be rendered *constructive*, allowing program extraction, in presence of classical proofs.

The idea is that the computational content of a proof (presented in a constructive or in a non-constructive manner) can be made explicit in order to build a new statement, which is *constructive* in the informal sense introduced above. Now we recover again the idea behind the program `nextprime`, and show how its statement can be rendered *constructive* (transferring to the statement the computational content on the proof), and then a certified program can be extracted from this statement. In order to make the statement *constructive*, several modifications have to be done.

In order to define a prime number bigger than a given natural number, we first define a primitive recursive function *some-prime-divisor-aux* which returns for each integer x, 0 or a prime divisor of x, which is known to return a value different from 0 thanks to Lemma 4.3.4 above.

primrec

[2] We are grateful to F. Sergeraert who attracted our attention to this important variant of constructivism.

```
some-prime-divisor-aux x 0        =  0
some-prime-divisor-aux x (Suc n)  =  (if (prime-cons (Suc n) ∧ dvd-cons (Suc n) x)
                                        then (Suc n)
                                        else (some-prime-divisor-aux x n))
```

Some comments can be made about this definition. First, the function has been defined in a recursive way. The advantages of using a primitive recursive function is that Isabelle offers facilities to prove properties of recursive functions, and also that code extraction is specially straightforward with recursive functions and types defined by induction. In a mathematical setting, it would have been more appropriate to define the function from top to down (starting from the pair (x, x) and going down to $(x, 0)$, which never would have been reached for $x > 0$). On the other hand, proving the properties of this definition would have been much more complicated, as far as induction cannot be used (at least in its primitive form).

In the definition of *some-prime-divisor-aux* it is also remarkable the use of both functions *dvd-cons* and *prime-cons*. Their behaviour have to be similar, respectively, to the usual division predicate in Isabelle, *dvd*, and to the Isabelle set *prime* (or to the predicate associated to it), distinguishing between prime and not prime numbers. The reason to redefine these functions is related to the strategy we have chosen. The constructive nature of the proofs can be avoided, relying then the proofs in classical logic, but on the contrary, the constructive content has to be transferred, to some extent, to the statements and definitions. As far as Isabelle defined predicates *dvd* and *prime* are based on existential quantifiers without a constructive definition, we cannot extract programs from these definitions. We explain this process of turning definitions into *constructive* definitions, and also the lemmas showing the equivalence between new and old definitions in the sequel.

First we detail the process for functions *dvd* and *dvd-cons*. Function *dvd-cons* will be just a redefinition of the Isabelle infix implemented function *dvd*, which given a pair of values m and n, of a type for which the product, the *times* operation, is defined, will be

```
consts
  dvd :: 'a :: times  => 'a => bool
defs
 dvd-def: m dvd  n  == ∃ k.  n  =  m ∗ k
```

Extracting code from this definition using Berghofer's tool is not possible. The presence of the existential quantifier in the definition makes impossible to extract directly a program from the predicate *dvd*. Therefore, we have to introduce a function called *dvd-cons*, equivalent to *dvd* but from which code can be extracted (in an informal sense, its constructive counterpart). Its definition is based on another recursive function, called *dvd-aux*, with three parameters. It decides whether the second parameter is the result of multiplying the first and the third ones. If so, then the first parameter divides the second one.

consts
dvd-aux:: *nat* => *nat* => *nat* => *bool*

primrec

dvd-aux a b 0	=	*False*
dvd-aux a b (*Suc n*)	=	(*if b* = *a* ∗ (*Suc n*) *then True else dvd-aux a b n*)

From the previous function, from which a program can be extracted in our setting, we will derive the definition of *dvd-cons*. The ML function obtained when we apply the code extraction tool to the Isabelle function *dvd-aux* is as follows

```
datatype nat = id0 | Suc of nat;

fun op__43_def0 id0 n = n
  | op__43_def0 (Suc m) n = op__43_def0 m (Suc n);

fun op__42_def0 id0 n = id0
  | op__42_def0 (Suc m) n = op__43_def0 n (op__42_def0 m n);

fun dvd_aux a b id0 = false
  | dvd_aux a b (Suc n) =
    (if (b = op__42_def0 a (Suc n)) then true else dvd_aux a b n);
```

The following facts can be commented on this piece of code. First, as far as in ML there is no primitive type for natural numbers (there is only a type `int` representing integers), the code piece defines a new type based on induction. The base element is named `id0`, and the constructor for the rest of elements is denoted as `Suc`. In order to define our function *dvd-aux* some previous functions are required. In this case, `op__43_def0` is the automatically assigned name for a recursive definition of addition over elements of type `nat`, as well as `op__42_def0` is a recursive definition of multiplication (where addition is used). Then, the ML definition of `dvd-aux` can be obtained in terms of `op__42_def0`.

We now define the predicate *dvd-cons* for every two natural numbers a and b to be true if *dvd-aux a b b* is true, or which is the same, if there is a natural number k smaller than b and such that $b = a * k$:

constdefs
dvd-cons:: *nat* => *nat* => *bool*
dvd-cons a b == (*dvd-aux a b b*)

From the Isabelle definition of *dvd*, based on the existential quantifier, we have had to derive a new definition called *dvd-cons*, with constructive content. The new definition seeks the natural number which multiplied by the first parameter is equal to the second one, instead of determining its existence by an existential argument. This definition allows the extraction of ML code from the Isabelle function *dvd-cons*.

```
fun dvd_cons a b = dvd_aux a b b;
```

The code is similar to the Isabelle definition, as expected.

It can be proved in Isabelle that our definition of *dvd-cons* is equivalent to the Isabelle definition of *dvd*. This is stated in the following lemma.

lemma *dvd-equiv-dvd-cons*: **assumes** $0 < a$ **and** $0 < b$ **shows** *a dvd b* = *dvd-cons a b*
proof –
 from *prems* **have** *one*: *a dvd b* $\Longrightarrow$ *dvd-cons a b*
  by (*simp only*: *dvd-impl-dvd-cons*)
 have *two*: *dvd-cons a b* $\Longrightarrow$ *a dvd b* **by** (*simp add*: *dvd-cons-impl-dvd*)
 from *one* **and** *two* **show** *?thesis* **by** *fast*
qed

If we avoid the special case where $a = 0$ and $b = 0$, that we did not consider in the definition of *dvd-aux*, both definitions are equivalent. (The case $a = 0$ and $b = 0$ could be considered just introducing a new conditional in the definition of *dvd-aux*.)

In the definition of *some-prime-divisor-aux* it also appears a function that we called *prime-cons*, because from the Isabelle definition of the *prime* set code cannot be directly extracted. The *prime* set is defined as follows.

constdefs
 prime :: *nat set*
 prime == { *p* . $1 < p \wedge \forall m.$ *m dvd p* --> $m = 1 \vee m = p$ }

As far as it includes an instance of the *dvd* predicate, which it is known to be no constructive (in the loose sense that no code can be extracted from it), it does not allow us to extract code from it, and a new definition must be supplied. A first solution that could be think of would consist in substituting the *dvd* predicate by its *constructive* counterpart *dvd-cons*; this solution showed to be not enough. Therefore, we decided to move to a setting we know compatible with the code extraction tool, based on recursive primitive functions. In the definition of *prime-aux*, a primitive recursive predicate based on our definition of *dvd-cons* is used.

consts
prime-aux :: *nat* => *nat* => *bool*

primrec
```
 prime-aux p 0         =  False
 prime-aux p (Suc n)   =  (if (p = 1) then False
                            else (if (n = 0 ∨ n = 1) then True
                              else (if (dvd-cons n p = True) then False
                                else prime-aux p n)))
```

The predicate only returns a *True* value for $n = 1$ and $n = 0$. The predicate has to be understood as follows: it should start from the value $Suc\ n = p$, and then recursive calls will be done backwards. If a value n is found such that $dvd\text{-}cons\ n\ p = True$ then p is not prime, and the predicate returns a *False* value, except in the cases where $n = 0$ or $n = 1$; in both cases, the (descendent) recursion has not found any divisor of p (apart from 1), and then the number is prime. In a similar way as we did with functions *dvd-aux* and *dvd-cons*, we now define a predicate based on *prime-aux* but with only one parameter, that later will be shown to be equivalent to the Isabelle *prime* set (the relationship between sets and predicates in Isabelle has been already pointed out in Section 1.3.2).

consts
prime-cons :: *nat* => *bool*
defs
prime-cons-def: *prime-cons x* == *prime-aux x x*

The previous definitions of *dvd-aux* and *dvd-cons* allow code extraction. The ML code obtained for these functions is the following:

```
val id1_def0 : nat = Suc id0;

fun prime_aux p id0 = false
  | prime_aux p (Suc n) =
    (if (p = id1_def0) then false
        else (if ((n = id0) orelse (n = id1_def0)) then true
             else (if (dvd_cons n p = true) then false
                  else prime_aux p n)));

fun prime_cons x = prime_aux x x;
```

As far as in our Isabelle definition of function *dvd-cons* the constant 1 can be found, ML also supplies a definition of it (`id1_def0`). Predicates `prime_aux` and `prime_cons` can be compared to their Isabelle definition.

Now it can be clarified why we defined the predicate *prime-aux* with $Suc\ n$ in the left hand side, and then in terms of n. The recursion had to be done backwards (resembling the definition of prime number as a number which does not have any divisor smaller than it, apart from 1). When we pretended to prove in Isabelle the equivalence of the definition of *prime-cons* with the one of *prime*, we found problems with the backwards recursion. Isabelle recursion principles are based on primitive (upwards) recursion. For instance, when we tried to prove the following property directly, we found several problems and even we were not able to complete its proof:

prime-aux-suc-impl-prime-aux:
$[\![\ 1 < k;\ \ k \leq p;\ \ prime\text{-}aux\ p\ p = True]\!] \Longrightarrow\ \ prime\text{-}aux\ p\ k = True$

From the definition of *prime-aux*, whenever the predicate returns the value *True* applied to a pair (p, p), it will return *True* also for every pair (p, k) with k smaller or equal than p (in other words, if there is no divisor of p smaller than p there will be no divisor of p smaller than k, with $k \leq p$). This property, that seems easy to prove by hand, becomes more complicated when we try to apply an induction argument inside of Isabelle. The proof introducing the induction principle is not compatible with our property, as far as the first one goes upwards and the second one, downwards. If the property is known to be verified for a value n, we cannot state it for the value *Suc* n, due to the nature of the definition. This situation leads us to seek a new property that can be derived by usual recursion from which we can prove the previous one. The following corollary showed to be useful in our case.

corollary *prime-aux-false-prime-p-false*:
$[\![\ 1 < k;\ \ \textit{prime-aux}\ p\ k = \textit{False};\ \ k \leq p]\!] \Longrightarrow \textit{prime-aux}\ p\ p = \textit{False}$

The previous property can be proved applying induction. If a number p has a divisor equal to or smaller than k, it will have a divisor smaller or equal than p with $k \leq p$. Applying induction on k, the previous corollary can be proved; from it, the proof of the result *prime-aux-suc-impl-prime-aux* is derived by contradiction (but not by induction, at least applied directly to the statement).

The following step would be to state the relationship between predicates *dvd-cons* and *prime-aux*. From the definition of the predicate *prime-aux*, if a pair (p, k) satisfies it, there will not be any number s smaller than k satisfying the predicate *dvd-cons* when applied to the pair (s, p). The proof is done in different parts. First, it is shown for the constructor *Suc* n, by the following lemma:

lemma *prime-aux-true-not-dvd-cons*:
$[\![\ 1 < k;\ \ \textit{prime-aux}\ p\ (\textit{Suc}\ k) = \textit{True}\ ]\!] \Longrightarrow \textit{dvd-cons}\ k\ p = \textit{False}$

Then we extend the previous result, instead of for *Suc* k, for every k such that $1 < k$ and $k < p$. Whenever the predicate *prime-aux* p p returns *True*, for every k it will be satisfied that *dvd-cons* k p = *False*, or, in other words, there will be no divisor of p apart from p and 1 (which is a result quite similar to the Isabelle definition of *prime*, which is the one we are seeking the equivalence with).

lemma *prime-aux-true-not-dvd-cons-generic*:
$[\![\ 1 < k\ ;\ k < p\ ;\ \textit{prime-aux}\ p\ p = \textit{True}]\!] \Longrightarrow \textit{dvd-cons}\ k\ p = \textit{False}$

Now, the following corollary shows that our predicate *prime-cons* is equivalent to the predicate associated to the Isabelle definition of *prime*

corollary *prime-cons-implies-prime*: **assumes** *prime-cons* p = *True* **shows** $p \in \textit{prime}$

Now it has to be shown that numbers belonging to the set *prime* verify the predicate *prime-aux*. Here induction can be applied in the natural way, and the following property is easy to be obtained:

lemma *prime-prime-aux-two*: **assumes** *k-g-1*: $1 < k$ **and** *k-l-p*: $k < p$ **and** $p \in prime$
  shows *prime-aux p* (*Suc 1*) = *prime-aux p k*

It is based on a previous result which ensures that for any number prime p, the value of *prime-aux p k* is equal to the result of *prime-aux p (Suc k)*, with $1 < k < p$. Joining together this result with lemma *prime-prime-aux-two*, we can conclude the following corollary.

corollary *prime-implies-prime-aux*: **assumes** $p \in prime$ **shows** *prime-aux p p* = *True*

Now that we have proved both equivalences between the Isabelle set *prime* and predicates *prime-cons* and *prime-aux*, we finally prove in the following corollary that the (non-constructive) Isabelle definition of the primes set can be identified with our constructive predicate.

corollary *prime-equiv-prime-cons*: $p \in prime$ = *prime-cons p*

Corollary *prime-equiv-prime-cons* and lemma *dvd-equiv-dvd-cons* allow us to establish the validity of the definition of function *some-prime-divisor-aux*. Invoking *some-prime-divisor-aux p p* a prime divisor of p will be found.

The ML code of this function is again quite similar to its Isabelle definition:

```
fun some_prime_divisor_aux x id0 = id0
  | some_prime_divisor_aux x (Suc n) =
    (if (prime_cons (Suc n) andalso dvd_cons (Suc n) x) then Suc n
      else some_prime_divisor_aux x n);
```

This function can be again converted into a function with a single argument, giving place to the function *some-prime-divisor*:

consts
some-prime-divisor :: *nat* => *nat*
defs
some-prime-divisor-def: *some-prime-divisor x* == *some-prime-divisor-aux x x*

Thanks to Lemma 4.3.4, and also to the following Isabelle lemma

lemma *some-prime-divisor-properties*: **assumes** *x-g-1*: $1 < x$
  shows *prime-cons* (*some-prime-divisor x*) $\wedge$ *dvd-cons* (*some-prime-divisor x*) *x*

function *some-prime-divisor* is known to return a prime divisor of the given parameter x. Moreover, this function is defined in such a way that the Isabelle code extraction tool can be applied to it, producing an ML function that satisfies the requirements we are looking for.

```
fun some_prime_divisor x = some_prime_divisor_aux x x;
```

If we take a closer look to the process that we have followed to obtain this ML function, it can be observed that, to some extent, we have had to define the program, giving constructive statements in Isabelle of the definitions of functions *prime-cons* and *dvd-cons* by using recursive functions.

consts
some-big-prime :: *nat* => *nat*
defs
some-big-prime-def: *some-big-prime* x == *some-prime-divisor* (x! + *1*)

The function called *some-big-prime* uses the Isabelle definition of the factorial and also the function *some-prime-divisor*. For any natural number x, it builds a number such that any prime divisor of it is greater than x, in this case $(x! + 1)$, and returns a prime divisor of this number. Once again, the code extraction tool can be applied to this function definition, obtaining the following ML function:

```
fun fact id0 = id1_def0
  | fact (Suc n) = op__42_def0 (fact n) (Suc n);

fun some_big_prime x = some_prime_divisor (op__43_def0 (fact x)
                                                       id1_def0);
```

First, a recursive definition of the factorial is extracted, based on the recursive definition of multiplication previously obtained (function `op__42_def0`). Then, the function `some-big-prime` is defined in terms of `fact` and also addition (function `op__43_def0`) .

Finally, with the following Isabelle theorem, we ensure that the value returned by this function, *some-big-prime* x, will be greater than x:

```
theorem x < (some-big-prime x)
proof (unfold some-big-prime-def)
  let ?p = some-prime-divisor (x! + 1)
  from some-prime-divisor-gt-zero
  have prime-cons: prime-cons ?p and dvd-cons: dvd-cons ?p (x! + 1)
    by (simp-all add: some-prime-divisor-properties)
  from prime-cons have prime-p: ?p ∈ prime
    by (simp add: prime-equiv-prime-cons)
  from dvd-cons have dvd: ?p dvd (x! + 1)
    by (simp add: dvd-cons-impl-dvd)
  show x < ?p
  proof −
    have ¬ ?p ≤ x
    proof
      assume ?p ≤ x
      with prime-g-zero and prime-p have ?p dvd x!
        by (simp add: dvd-factorial)
      with dvd have ?p dvd (x! + 1) − x! by (rule dvd-diff)
      then have ?p dvd 1 by simp
      with prime-p show False using prime-nd-one by auto
    qed
    then show ?thesis by simp
 qed
qed
```

This proof, a variant of Euclid's argument, is an adaption of a proof script presented in [Wenzel and Wiedijk, 2002]. It has been made using a "reductio ad absurdum" strategy. A constant *?p* is fixed, defined to be equal to *some-prime-divisor* $(x! + 1)$. Then, we try to prove that *?p* is smaller than or equal to x, arriving at a contradiction, since x would divide to both $x!$ and $x! + 1$. The proof has a non-constructive presentation. It might be considered unsuitable from a constructivist's point of view, but it is well-known and easily understood. The relevant fact about it is that, no matter the logic underlying the proof, the *statement*, as well as the functions used in the statement, are constructive, and Berghofer's tool can be applied to the complete Isabelle theory file to obtain the following ML program, some of which fragments have been already presented[3]:

[3]The code extraction tool assigns names to its own functions and constants automatically; here we have changed some of these names in order to make them more meaningful.

```
structure Generated = struct

datatype nat = id0 | Suc of nat;

fun sum id0 n = n
  | sum (Suc m) n = sum m (Suc n);

fun times id0 n = id0
  | times (Suc m) n = sum n (times m n);

fun dvd_aux a b id0 = false
  | dvd_aux a b (Suc n) =
    (if (b = times a (Suc n)) then true else dvd_aux a b n);

fun dvd_cons a b = dvd_aux a b b;

val id1 : nat = Suc id0;

fun prime_aux p id0 = false
  | prime_aux p (Suc n) =
    (if (p = id1) then false
      else (if ((n = id0) orelse (n = id1)) then true
        else (if (dvd_cons n p = true) then false else prime_aux p n)));

fun prime_cons x = prime_aux x x;

fun prime_number n = ((fn p => (prime_cons p andalso dvd_cons p
n)));

fun some_prime_divisor_aux x id0 = id0
  | some_prime_divisor_aux x (Suc n) =
    (if (prime_cons (Suc n) andalso dvd_cons (Suc n) x) then Suc n
      else some_prime_divisor_aux x n);

fun some_prime_divisor x = some_prime_divisor_aux x x;

fun fact id0 = id1
  | fact (Suc n) = times (fact n) (Suc n);

fun some_big_prime x = some_prime_divisor (sum (fact x) id1);
```

Figure 4.2: ML extracted program

Some facts can be pointed out about this code, apart from the ones already mentioned. Here we have replaced the automatically assigned names to the recursive functions performing addition and multiplication by `sum` and `times`, which clarifies the content of the operations. This ML code is to be compared with the Common Lisp program `nextprime`. Both functions compute a prime number greater than its argument x. Of course, `nextprime` is quite more efficient. We made some tests with the ML obtained program for function *some-big-prime*, and we got the expected results just for values below 5 (i.e., `Suc (Suc (Suc (Suc (Suc id0)))))`). For values greater than 5, the program showed to be too much time consuming. Thanks can be given here to S. Berghofer, which kindly explained us the process to identify the Isabelle type *nat* with a restriction of the `integer` type of ML, optimizing the programs performance. These changes required a posterior version of Isabelle than Isabelle2004, the one used for our work, and we decided not to introduce them here. The improvements suggested by Berghofer would increase the applicability of the program. In any case, the Isabelle theorem stating that $x <$ *some-big-prime* x, together with the program extracted of the Isabelle definition of function *some-big-prime*, satisfy the requirements that we have imposed of finding a certified program that, for every natural number, is capable of building a greater prime number. This goal has been achieved, as we required, even when the Isabelle proof of the theorem stating $x <$ *some-big-prime* x was made in a non-constructive way (by using a "reductio ad absurdum" argument).

4.4 Application to Computer Algebra

When applying these ideas to the Kenzo program, the first observation that can be made is that the elaboration done in the previous section is unnecessary: many of the theorems to be proved already have a *constructive statement* (or can be easily transformed into such a statement), in the sense that the statements contain the computational information needed to define the goal. If we now take a closer look at the collection of lemmas in Section 2.2.2, it can be observed how in the statements, a new object is defined from which some property has to be stated. As far as the new object is defined, and its definition is given explicitly, the statement can be considered constructive. Therefore, the same work already made for the formalization that we have described in Chapter 3, can be reused for extracting code from statements. For instance, in the statement of Lemma 2.2.14, a reduction is defined from a chain complex (D_*, d_{D_*}) and an endomorphism h satisfying certain premises; the reduction goes from (D_*, d_{D_*}) to a chain subcomplex of it, defined as $(\ker p, d_{D_*})$, with $p = d_{D_*}h + hd_{D_*}$ and h such that $hh = 0_{D_*}$ and $hd_{D_*}h = h$. Contrarily to the previous example about prime numbers, where we had to develop the whole process of construction of the candidate to satisfy the lemma's requirements (i.e., the new prime number), the statement itself proposes the reduction between the chain complex and its subcomplex. In Lemma 2.2.14, this reduction is defined as $(\mathrm{id}_{D_*} - p, \mathrm{inc}_{\ker p}, h)$. A similar situation can be observed in Lemma 2.2.11. There, it must be proved that the chain complex (C_*, d_{C_*}) and the chain subcomplex $(\mathrm{im}\, gf, d_{D_*})$ are isomorphic, where the tuple (f, g, h) is a reduction between (D_*, d_{D_*})

and (C_*, d_{C_*}). Again, the isomorphism is explicit and provided in the statement.

As explained at the end of Section 4.2, we consider a simplified version of the Kenzo composition. More concretely, algebraic structures are considered ungraded (i.e., the degree of homomorphisms is 0) and, in addition, groups are considered not to be free (that is to say, the strategy is always `:cmbn`, *by combination*, following Kenzo terminology). From the different approaches we introduced in Chapter 3, we have chosen here the morphism based one (see Section 3.7), where homomorphisms were represented through records. The first reason is that we found it to be the more appropriate among the different approaches presented from the point of view of accuracy with respect to the mathematical setting. The second one is the similarities that it keeps with the Kenzo implementation. In this approach, composition of group homomorphisms was defined as follows:

```
constdefs
group-mrp-comp  ::  [ ('b, 'c, 'e, 'f) group-mrp-type, ('a, 'b, 'd, 'e) group-mrp-type]
                     => ('a, 'c, 'd, 'f) group-mrp-type
group-mrp-comp g f  ==  (| src = src f, trg = trg g, morph = (morph g) ∘ (morph f),
                        src-comm-gr = src-comm-gr f, trg-comm-gr = trg-comm-gr g|)
```

The composition of two homomorphisms f and g represented through records is defined to be a new record, where the source fields *src* and *src-comm-gr* are obtained from f, the target fields *trg* and *trg-comm-gr* are taken from g, and the field storing the functional information is defined as the composition *(morph g)* $\circ$ *(morph f)*. Composition defined through this function has to be proved to be again an object satisfying the axioms of the set *group-mrp* (these axioms were already introduced in Section 3.7.3). Here we introduce one of the lemmas proving that the composition of two objects *group-mrp* is again an object *group-mrp*, under some determined premises over the source and target algebraic structures:

```
lemma group-mrp2-composition:
  assumes A1: group-mrp2 A
  and B1: group-mrp2 B
  and C1: trg-comm-gr A = src-comm-gr B
  and D1: trg A = src B
  shows group-mrp2 (B ∘ A)
proof (unfold group-mrp-comp-def group-mrp2-def, intro conjI, simp-all)
  from prems show diff-group (src A) by (simp add: group-mrp2-def)
  from prems show diff-group (src-comm-gr A) by (simp add: group-mrp2-def)
  from prems show sub-diff-group2 (src A) (src-comm-gr A) by (simp add: group-mrp2-def)
  from prems show diff-group (trg B) by (simp add: group-mrp2-def)
  from prems show diff-group (trg-comm-gr B) by (simp add: group-mrp2-def)
  from prems show sub-diff-group2 (trg B) (trg-comm-gr B) by (simp add: group-mrp2-def)
  from prems show (morph B o morph A) `(carrier (src A)) ⊆ carrier (trg B) by (unfold
group-mrp2-def image-def, auto)
  from prems show morph B ∘ morph A ∈ hom-complection (src-comm-gr A) (trg-comm-gr
B)
```

proof (*unfold hom-complection-def group-mrp-def*
complection-fun2-def complection-def, *simp*, *intro conjI*)
assume *group-mrp2 A* **and** *group-mrp2 B* **and** *trg-comm-gr A* = *src-comm-gr B*
then show *EX g*. *morph B o morph A* = (*%x*. *if x* : *carrier* (*src-comm-gr A*) *then g x else one* (*trg-comm-gr B*))
proof (*intro exI* [*of* - *morph B* ∘ *morph A*], *auto simp add*: *expand-fun-eq group-mrp2-def*)
fix *x* **assume** *x* ∉ *carrier* (*src-comm-gr A*)
and *morph A* : *hom-complection* (*src-comm-gr A*) (*src-comm-gr B*)
and *morph B* : *hom-complection* (*src-comm-gr B*) (*trg-comm-gr B*)
show *morph B* (*morph A x*) = *one* (*trg-comm-gr B*)
proof−
from *prems* **have** *morph A x* = *one* (*src-comm-gr B*) **by** (*intro hom-complection-closed2*)

then have *morph B* (*morph A x*) = *morph B* (*one* (*src-comm-gr B*)) **by** *simp*
also from *prems* **have** ... = *one* (*trg-comm-gr B*)
by (*intro group-hom.hom-one* [*of* - - *morph B*],
unfold group-hom-def group-hom-axioms-def group-mrp2-def diff-group-def
hom-complection-def, *simp*)
finally show *?thesis* **by** *simp*
qed
qed
next
from *prems* **show** *morph B* ∘ *morph A* ∈ *hom* (*src-comm-gr A*) (*trg-comm-gr B*)
by (*unfold hom-def*, *simp*, *intro conjI*, *unfold Pi-def group-mrp2-def hom-complection-def hom-def*, *simp*)
(*auto simp add*: *magma.m-closed*)
qed
qed

This Isabelle lemma ensures that composition, as defined, is closed. We can now apply Berghofer's extraction tool to the Isabelle definition *group-mrp-comp*, obtaining the following ML program (only the most relevant part is shown here):

```
datatype unit = Unity;

fun comp g f = (fn x => g (f x));

fun group_mrp_comp g f =
  (src f,
    (trg g,
      (comp (morph g) (morph f),
        (src_comm_gr f,
          (trg_comm_gr g, Unity)))));
```

As it can be observed, as far as our definition of composition is constructive in the sense that it fulfills the definition of the new homomorphism, the extracted ML program resembles the Isabelle definition.

The proof of the previous Isabelle lemma *group-mrp2-composition* is to be considered as a *certificate of correctness* of the ML program (assuming, as usual, the soundness of Berghofer's translation). This (certified correct) ML program can be now compared with the *real* corresponding Kenzo program or better, to ease the reading, with the simplified Common Lisp version given at the end of Section 4.2 and presented here again:

```
(defun CMPS (g f)
  (build-mrph :sorc (sorc f) :trgt (trgt g)
              :intr #'(lambda (cmbn)
                         (cmbn-? g (cmbn-? f cmbn)))))
```

Both functions `group_mrp_comp` and `CMPS` are quite similar, as expected. One of the differences is due to the special implementation that we have used in Isabelle for homomorphisms, keeping record of both their source and also their domain of definition. Therefore, the composition in ML also must store both structures in different fields. This is the only difference between the ML and the Common Lisp functions. The composition of functions in ML has been extracted in the form of a new function (`comp`), whereas in the Kenzo implementation was done in a single step, but with a similar definition. The rest of differences depend just on the special programming languages syntax.

In summary, in this chapter we have applied Berghofer's extraction tool for Isabelle scripts to obtain Computer Algebra programs with a certificate of correctness. The main idea consists in extracting code from *statements* and not from *proofs*, as it is usually done in constructive type theory. From a technical point of view, it would be more accurate to say that code is extracted from *definitions* appearing in statements, but it has been considered more appealing to exploit the couple statement/proof. In fact, in the arithmetic example in Section 4.3, it is clear that we are *programming* inside of Isabelle, by transforming Isabelle defined predicates, functions and sets into recursive functions, and proving, at the same time, the correctness of these transformations.

Even if it seems that in the elementary examples from Computer Algebra in Algebraic Topology this work of *programming in Isabelle* is not necessary, as far as the definitions we have provided are already *constructive* (composition of homomorphisms, for instance), important challenges are still open. It would be necessary to bridge the gap between the ML and Common Lisp programming languages, and even more difficult, the gap between the ML programs extracted (that could be very inefficient) and the corresponding performing programs which are really usable. With respect to this, the toy example with prime numbers could illustrate the strong difficulties (in terms of proving efforts within Isabelle) to obtain a reasonably efficient program. Thus, finally it will be perhaps unavoidable to *program in Isabelle* in order to get usable programs. From our point of view, this problem of using proof assistants to synthesize "real-life" programs is a central one in intelligent information processing (see, for instance, [Schirmer, 2005] and [Dybjer et al., 2004]).

Chapter 5

Conclusions and Further Work

5.1 Conclusions

This work should be understood as a first step towards mechanized reasoning in Algebraic Topology, with an emphasis on its application to prove the correctness of Computer Algebra programs. More specifically, our inspiration comes from the system Kenzo [Dousson et al., 1999], and from previous work done to specify this system [Lambán et al., 2003].

We have focused on a concrete result, called Basic Perturbation Lemma (or BPL, in short). After briefly explaining the relevance of the BPL in the field of algorithmic Algebraic Topology, a very detailed proof of the BPL has been shown. This proof follows the guidelines of a proof originally presented in [Rubio and Sergeraert, 1997]. It is interesting to note that in [Rubio and Sergeraert, 1997] around 70 lines were employed to describe the proof (including the statement and proof of an auxiliary lemma), while our presentation needs around 20 pages. This observation should be examined to the light of the three items mentioned in the introduction:

1. the *idea* underlying a proof;
2. the presentation of the proof in a mathematical text;
3. the very proof in the sense of symbolic logic.

Our proof of the BPL shows how one same idea (point 1) admits very different presentations (point 2). Chapter 3 in the memoir illustrates clearly that even in cases of exaggeratedly detailed proofs, a complete proof in the symbolic logic sense (point 3) continues to be dramatically distant.

Two main aspects have been detected in our proof (when comparing it with the original proof in [Rubio and Sergeraert, 1997]) influencing the length of proof presentation.

On the one hand, the fact that, when reasoning with chain complexes, several kinds of morphisms can be considered, depending on how they respect the degrees and if they are coherent or not with respect to the differentials. In extreme cases, one same map can be considered as a graded group morphism or as a chain complex morphism in a small proof fragment. On the other hand, it is quite usual to restrict a morphism to some subgroup (or to some chain subcomplex) of its source, and to refer to the initial morphism and to the corresponding restriction in an undistinguished manner.

Both aspects can be considered particular cases of another issue remarked in the introduction: the great freedom to name entities in standard mathematical texts. A same variable (even a same letter) can denote different objects (different but intimately related in a conceptual sense, of course) in a same proof fragment.

This question of variables seems difficult to be tackled as a whole (it is so general that, in these terms, it is rather an ill-posed problem). Nevertheless, the two aforementioned particular cases have been treated in our work. The first one in, let us say, an *internal* way inside Isabelle; the second one in a manner which could be considered *external* to Isabelle.

The first aspect is alleviated, in part, by using a general resource of Isabelle/HOL: *extensible records*. This datatype provides Isabelle with a kind of *inheritance*. Thus, a chain complex can be also considered simply as a graded group by the type system, and then morphisms are more easily treated from and to different structures underlying the same object.

For the second problem, that of restriction, our solution has been to represent a morphism as a more complex data structure. That is to say, we have changed our way of encoding morphisms. This is why we consider this proposal as *external* to Isabelle: it does not depend on an Isabelle resource, but in a new conceptual scheme. Morphisms are represented as triples: in addition to the functional information (which includes its source and target objects), the actual source and target spaces are also incorporated. Therefore, a same mapping can be used in many different morphisms, emulating the way of working of mathematicians with respect to restrictions (and extensions) of functions.

This technique of changing our *perspective* on a given mathematical object (in the just evoked example: going from a morphism represented as a map to a morphism as a triple), is a central point in Chapter 3, where really starts the encoding of the proof of the BPL in Isabelle. We have developed four different approaches to try to solve the problems for a very small fragment of the proof. It turns out that the notion of *abstraction*, introduced by Hoare to study the implementation of data structures [Hoare, 1972], allows us to analyze the different approaches, and, even more, to direct our research in some delicate points.

Even if the discussion is carried out on a small fragment of the proof, we claim that some of the problems raised (and solved) are of a quite general nature, since related to the generic reasoning on algebraic structures and algebraic morphisms. (Another major simplification in Chapter 3 is the adoption of an *ungraded* version of the BPL;

nevertheless we consider this simplification rather as a technical issue, trying to get a separation of concerns: the problems of working with graded structures are orthogonal to the central questions around implementing a proof of the BPL in a theorem prover.)

To understand the several approaches we have tried, it is necessary to note that the BPL proof (as very detailed in Chapter 2) deals with *morphisms*. Then, a morphism can be viewed in two different ways:

- as element of an algebraic structure (the endomorphisms ring, for instance);
- as a set-theoretic mapping (satisfying certain properties).

It is clear that the first view is incomplete, since our interest is related to the algorithmic use of morphisms (technically, it can be said this representation of a morphism lacks *computational content*). But, it is clear also that forgetting that first perspective avoids reasoning with morphism in a level of abstraction typical in standard mathematical texts. Our three first approaches explore this distinction (the fourth one being a rewording of the third approach, by using a technical tool from Isabelle, namely *locales*).

In the first approach, named by us *symbolic approach*, only the first interpretation of a morphism is considered: it is represented by means of a generic element (a *symbol*) of a generic ring. If a property relevant to the BPL can be expressed in this context, a very efficient way of reasoning is achieved. We have named it as *equational reasoning*: proofs are carried out through rewriting of expressions, applying the axioms of generic algebraic structures (as rings or groups). This allows us a high degree of automation, by using the Isabelle tactic *algebra* (built by C. Ballarin). However, the expressiveness of this approach is low (there are fragments of the BPL proof which are related to morphisms as mappings; for instance when it is necessary to restrict a morphisms to a distinguished subset of its source) and, as previously mentioned, it lacks computational content.

Thus, new ideas are needed. In our second approach, we consider a morphism in the second sense evoked above: a set-theoretic mapping, satisfying certain properties. This has been called, not surprisingly, *set-theoretic approach*. This point of view is the orthodox one, in a naive sense: from set theory, a hierarchy of algebraic structures is built, as usual in Universal Algebra. We have tried to emulate this way of working in Isabelle, starting from the most elementary structures (magmas, semigroups, monoids, ...). The main technical tool to this aim has been *extensible records*, which enable Isabelle with some inheritance features. This allows us also to deal with the different kind of morphisms which can be defined on a same algebraic structure, as has been commented above.

Despite of its soundness and of the Isabelle technical help, it becomes evident early that proofs become unfeasible in this setting: going down always to reason with the *elements* of each algebraic structure (instead of working with the structure itself, and with its properties as a whole), increases enormously the length of the proofs, and, even worse, hides the important logical steps among plenty of boring technical details. The conclusion is a confirmation of something well-known: mathematicians refer to the low-level set-theoretic machinery only to establish the basics; then, they work with higher reasoning level.

The third approach tries to encompass the advantages of the two previous approaches, avoiding its drawbacks. In other words, we have tried to recover equational reasoning (and thus a great level of automation) without loosing expressiveness and computational content. To this aim, it has been necessary to change in deep the representation of morphisms, both from an operational and from an structural point of view.

From the first perspective, it has been necessary to deal with partiality, in order to get a representation where *equals* in the abstract model (that is to say, equals in Mathematics) come from *equals* in computer memoir (that is, as Isabelle objects). Isabelle only admits *total* functions and morphisms in Homological Algebra are also usually *total* maps. Then, where *partiality* appears? Partial maps appears due to the following reason. Isabelle functions are total on *types*. But morphisms are not defined on types, but on *sets* acting as *carriers* of algebraic structures. Therefore, when representing a morphism in Isabelle only the data of the type belonging to the carrier have a well-defined image. Each datum in the complement of the carrier *must* have an image (functions in Isabelle being total), but Mathematics says nothing about that image, which becomes *arbitrary*. Thus, when we want to use in Isabelle an equality between two morphisms (and it is mandatory if equational reasoning must work), we find that the system is not capable of deciding, due to the *arbitrary* values of the Isabelle function in the complement of the carrier set. Our solution consists in sending each datum in the complement to a distinguished element (the unit) of the target algebraic structure. This *completion* of the function implies that two equal morphisms in the abstract model are represented in the computer by two objects which are recognized to be equal by Isabelle (putting it in technical terminology, the representation is *faithful*: the abstraction function is injective). So, equational reasoning is, in principle, possible.

But, from a structural point of view, we face the problem of variables mentioned above: the BPL proof uses intensively that a same mapping can define different morphisms by changing its source or target. These entities are usually denoted by the same variable, allowing the user shorter and cleaner proofs. To get this higher abstraction level in Isabelle, we changed one more time our representation for morphisms. Now, a (completion of a) function is enclosed with a source and a target. These triples allow us to pass from a morphism to one of its restriction (for instance), without rebuilding the whole structure: only a simple field replacement is required.

Once these operational and structural issues are enabled, it is necessary to prove that the set of endomorphisms (represented as triples) in Isabelle is endowed with the corresponding ring structure. As it is no surprise, it turns out that the number of code lines to prove that theorem in Isabelle is quite considerable, being larger than in the proofs developed in the second approach (which were qualified as “unfeasible”!). This is quite natural: the load work for an *infrastructure* can be assumed, due to the important reuse which can be implemented on it. In our case, in particular, equational reasoning is fully recovered with this approach, including the automation provided by the *algebra* tactic.

Finally, the fourth approach, called *interpreting approach*, is not of the same nature as the previous ones. The reason is that the quality improvement obtained with it is due to an *internal* Isabelle feature, namely *locales*. Interpretation of locales allows the user some important notational commodities: several instances of a same algebraic structure (several rings, for instance) can be used in the same Isabelle fragment, importing every property already proved on it. Even if expressiveness is not empowered essentially in our context, the commodities introduced by locales can imply an improvement in the readability and the construction of proofs which can not be underestimated.

In Chapter 4, our results have a twofold interest. On the one hand, they allow us to get closer to our initial problem of proving the correctness of the Kenzo system. On the other hand, they imply a search on the relationship of our work with constructivism. On the contrary, other approaches related to code extraction are not dealt with. In particular, the efficiency and generality of the programs extracted are not studied, due, essentially, to the limitations of the current technology.

The main tool in Chapter 4 is S. Berghofer's program to extract ML code from Isabelle scripts ([Berghofer, 2003a, Berghofer, 2003b, Berghofer, 2004]). Due to the richness of higher order logic, it is clear that not every fragment from Isabelle can be used to extract code from it. This poses interesting questions on the limitations of Berghofer's tool and also on its comparison with other systems based on constructive mathematics (Coq [Bertot and Castéran, 2004] is the closer alternative). Instead of tackling these problems with a *theoretic* spirit, we were inspired from a *pragmatic* aim: to study the application of Berghofer's program to algorithms from Homological Algebra. It turns out that most of the interesting theorems (as documented in Chapter 2) have a remarkable property: in the statement an object (algebraic structure, morphism, reduction and so on) is defined, and some property of it is asserted. We called this kind of statement *constructive statement*. The key observation then is that code can be extracted from this statement (or, to be more precise, from the definition introduced in the statement), without any reference to its proof. Since the proof is not relevant for the code extracted (except from the essential question of ensuring its correctness, of course), the logic used in that proof is uncoupled from the extraction mechanism. In particular, the proof could be *non-constructive*.

In order to reduce the complexity of the code and the proofs, we explore this delicate problem with a more elementary example, by means of a simple Common Lisp program which computes the next prime to a natural number given as argument. Even if simple, this example is not trivial at all: it is the same example Markov's chose ([Markov, 1971]) to illustrate his generalization of constructive mathematics (known nowadays as *recursive constructive mathematics*). Instead of trying to prove the correctness of this program by means of Isabelle (apart from the theoretical difficulties related to constructivism, we would be faced to the technical difficulty of linking ML and Common Lisp), we have reworded the problem to establish it as a constructive statement. Then an ML program equivalent to the initial one is extracted with the help of Berghofer's tool, while the proof of the theorem is done by using a "reductio ad absurdum" argument, giving a non-constructive flavour to it. All these results, quite expensive in number of Isabelle code lines, amounts to *programming* in Isabelle (in addition to the usual tasks of *specifying* and *proving* in Isabelle). The previous ideas were then applied to a simple case in Homological Algebra: the composition of two morphisms.

In summary, the memoir contains several thoughtful proposals both for mechanizing reasoning in Homological Algebra and for obtaining certified Computer Algebra programs in this field. In particular, we have stressed the importance of the abstraction process in mechanized mathematics, and introduced several conceptual schemes which could be reused in many similar areas of application. Another product of our research is to make clear the limitations of certain approaches, and to establish the difficult places in the rest of work to be done. It is clear that the research is not ended: to ensure that our proposals are really reusable is necessary to advance in the BPL proof, and also to apply the extraction methods to more complex fragments of the algorithms. We explain our future projects in the next (and last) section.

5.2 Further Work

First, it would be necessary to compare our contributions with other possibilities. And this, not with respect to other alternatives on the same problem (our work is, in this sense, quite new and original, in our humble opinion), but rather to other concurrent tools. The most convenient seems to be to translate our methods to the system Coq [Bertot and Castéran, 2004]. To this aim, it could be interesting to use the system FOCAL [Foc, 2005], which provides an integrated environment for specifying, programming (linked to the programming language OCAML [Leroy et al., 2005, Boulmé et al., 1999]) and proving (linked to Coq).

But the most evident work to be continued is to finish the second part of the BPL proof. We think that our proposals will be sufficient to this task. Perhaps the more important open question is to know if our handling of the *equality problem* is enough. Up to now, the equality between morphisms has been treated (by means of the *completion* notion) in such a way that it has been unnecessary to *overload* the equality symbol in Isabelle. In situations where the equality extends to algebraic structures as a whole, perhaps this overloading or another technical or conceptual idea could be required.

Then the implementation the *first part* of the proof should be undertaken. It deals with a *functional series*, and it is clear that some kind of inductive process should be introduced to handle this infinite structure. Although Isabelle supports this kind of reasoning, some open questions are expected, related both to code extraction and to foundation aspects.

Effectively, the extraction of ML code from routine proofs in Homological Algebra (as those appearing in the second part of the BPL proof) seems to be solved by means of Berghofer's tool and our technique of *constructive statements*. However, the series introduces problems on termination, and it is more difficult, without more research, to foresee if current technology will be enough. Let us explain this point with a little more detail.

The proofs in Chapter 2 related to the series are based in the fact that the functional formal series defines a morphism, because when applied over each element it becomes a *finite* sum. This depends on the notion of *locally nilpotent* endomorphism. In the version presented in this memoir (the usual in the literature) the situation is very similar to the arithmetical example considered in Chapter 4: it is known that a bound exists, but no information is explicitly given on the range of that bound. Thus, to prove termination is problematic and would need Markov's principle, going beyond strict constructivism (and thus, perhaps, beyond of Berghofer's tool capabilities). Nevertheless, from a pragmatic point of view, the definition can be transformed into a constructive one (giving rise to the notion of *locally constructive nilpotency*) by requiring an explicit function computing the degree of nilpotency for each element. This new definition is applicable to any instance of the BPL needed in Kenzo (and, to our knowledge, in any application in algorithmic Homological Algebra or Algebraic Topology), showing a way for code extraction in this setting (no so different of our treatment in Chapter 4 for the prime numbers example).

This small discussion signals clearly that the problem of the foundation nature of Algebraic Topology (if it can be rendered strictly constructivist, recursive constructivist or is simply non-constructive) remains open (this question has been raised in several editions of the *Mathematics, Algorithms, Proofs* conferences; see a brief note on these issues at MAP2005 in [Rubio, 2005]).

Even if any practical or theoretical problem is solved, and ML code is safely extracted from the BPL modeled in Isabelle, there will be yet the gap from ML to Common Lisp (the programming language in which Kenzo is written). Several approaches are here possible. One of them is to extract code from Isabelle to Common Lisp. The other one is to write a (certified!) transformer from ML into Common Lisp. Both projects seem to deserve general attention, beyond our particular context of application.

And going one more step ahead, even if Common Lisp code can be extracted, the problem of generating code similar to *Kenzo* will remain still open. Putting it in other words, the problem of extracting efficient code (comparable to Kenzo) does not seem affordable in the state of current technology. This is the reason why our approach to this problem is very similar to the one showed in this memoir with respect to the mechanized proof of the BPL. The interest is not to get a *complete* automated proof, but to increase our knowledge (on the very proof of the BPL, on the limits of the current mechanized reasoning systems, or, in short, on the tight interplay between Mathematics and Computer Science). In this same vein, the real goal is not to prove in an automated way the correctness of Kenzo (which is a quite secure system, after several years of successful *testing*), but to find out methods to increase the reliability of software systems, without losing efficiency or usability. Automated theorem provers can be instrumental for these purposes. A small step in this direction can be understood as the main contribution of this memoir.

Appendix A

Isabelle Files

This appendix contains some information referred to the Isabelle files which were developed to write down Chapters 3 and 4.

A.1 Mechanizing the proof of the BPL

The Isabelle files employed in Chapter 3 have been divided in four groups, as the four approaches that are described there. The proofs that were exposed in the *symbolic approach* in Section 3.5, as well as some other examples can be found in file *"symbolic/symbolic.thy"*. The proofs found there are based on equational reasoning, but just some basic properties can be proved.

The theory files proving Lemma 2.2.11, using the tools introduced in Section 3.6, can be found in folder *"set_theoretic/lemma1"*; proofs have been done following the ideas exposed in what we called the *set theoretic approach*. Five files are given. In file *"lemma1_previous.thy"* abelian groups are introduced, as well as homomorphisms between them. Then, some introductory lemmas are proved; finally, the composition of abelian group homomorphisms is proved to be closed. In file *"lemma1_isom_sets.thy"*, a version of Lemma 2.2.11 is given in a simplified version: differential groups are substituted for sets, and isomorphisms for bijections. Then, in file *"lemma1_isom_groups.thy"*, sets are enriched to obtain groups, and a definition of isomorphic groups is given. Then, the same lemma is proved in a version for groups. In a similar way, in the file *"lemma1_isom_ab_groups.thy"*, groups are enriched to obtain abelian groups, and the lemma is proved for abelian groups, making use of the previous version (thanks to extensible records, presented in Section 1.3.2). In file *"lemma1_isom_cc.thy"*, differential groups are defined, as well as homomorphisms between differential groups. Two main lemmas are introduced in this file. The first one states that under the premises of Lemma 2.2.11, an isomorphism between the underlying abelian groups can be explicitly defined; its proof is based on the previous version, and is almost direct. Then, the same lemma is proved, stating that the isomorphism can be seen also as an isomorphism between differential groups, which corresponds to Lemma 2.2.11.

The files in folder *"set_theoretic/lemma2"* present an unsuccessful attempt of proving Lemma 2.2.14 again with the tools introduced in the set theoretic approach. Nevertheless, some of the definitions and lemmas proved there are relevant by themselves. In file *"Kernel.thy"* the definition of kernel of a homomorphism is introduced. In addition to this, a lemma stating that the kernel of a homomorphism with the inherited operations from the source algebraic structure, satisfies also the axioms of such an algebraic structure is proved (for instance, the kernel of a group, with these operations, is a subgroup). The same lemma is proved for groups, abelian subgroups and differential subgroups. The version for differential subgroups corresponds to Proposition 2.2.9. In file *"Reduction.thy"* the definition of reduction is introduced. With the tools available, the definition of reduction is not very easy to be used. Two main lemmas are then proved. One stating that a combination of the differential and the homotopy operator is again a chain complex isomorphism. Such a simple fact, which corresponds to the first part of the proof of Proposition 2.2.12, required a great number of lines. The second one corresponds exactly to Proposition 2.2.12, which thanks to the previous lemmas already proved in Isabelle was proved in quite an easy way. Finally, in file *"lemma2_is_reduct.thy"*, some basic facts about reductions are proved; Lemma 2.2.14 is also stated, but we were unable to produce a proof of it in a comfortable way. Thus, we decided to introduce new tools, in what we called the *morphism based approach.*

The files related to the morphism based approach exposed in Section 3.7 can be found in folder *"triples"*. File *"basic_def_2.thy"* is devoted mainly to the introduction of new concepts. First, completions are defined and proved to be closed under composition. Then we also introduce the definition of homomorphisms as tuples storing information about their source and target. With this new definition we prove that composition is closed under diverse circumstances; for instance, when the target of one of the homomorphisms is included in the source of the second one. In file *"MRP_equiv.thy"* a relation is defined between triples. This relation is proved to be an equivalence relation. As stated in Section 3.6.5, trying to define the equivalence classes and dealing with them could be an interesting problem. Also some of the different versions of Lemma 3.7.1 are proved, in order to be used in later proofs. File *"group_mrp_ring.thy"* is mainly devoted to proofs. First it is proved that tuples including completion homomorphisms form a ring. Then some proofs are implemented with tuples in an equational way. Finally, and applying some of the versions of Lemma 3.7.1, together with *equational reasoning*, the properties in the statement of Lemma 2.2.14 are proved (avoiding some of the problems previously found in the former approaches).

The files where the ideas in Section 3.8 are implemented, can be found in folder *"inst_locales"*. Files *"HomGroup.thy"* and *"HomGroupRestrict2.thy"* reveal the limitations of the Isabelle definitions of homomorphisms and restricted functions to work with them in an equational way, and moreover, to implement with them algebraic structures with the usual operations. Then, in file *"HomGroupComplection.thy"* we introduce the definition of completion functions and some facts that allow us to prove that the set of endomorphisms of a commutative group, with the suitable operations, form a ring.

Then, in file *"HomGroupsComplection.thy"* we prove that the set of homomorphisms between two commutative groups form also a commutative group. With these previous lemmas, in file *"lemma2_properties_1_to_4.thy"* we can define a *locale* representing the mathematical setting of Lemma 2.2.14, and we can also implement its complete proof.

A.2 Extracting Computer Algebra Programs from Statements

The files where we have implemented the ideas and examples exposed in Chapter 4 can be found in folder *"ProgramExtraction/Primes"*. In file *"Monoid.thy"* it can be found the first examples in Chapter 4.3. In first place, we define an operation constructing a monoid and we give an Isabelle proof of Lemma 4.3.1. Then we also define a record with the set of integers, usual addition as *mult* field and 0 as *unit*. We prove this record to satisfy the monoid axioms. Finally, by using the code extraction tool over these definitions, we obtain an ML program which can be found in file *"Monoid.ML"*. In file *"Prime_number_theorem.thy"* it can be found a complete proof of the Euclid's lemma using the Isabelle definitions of division and prime number. These definitions are showed to be incompatible with the code extraction tool, due to the presence of existential quantifiers. Then, new definitions are proposed for both functions; these definitions are proved to be equivalent to the Isabelle ones. Then, a new proof of Euclid's Lemma is given with these definitions. Finally, an ML program is extracted from them (see file *"Prime_number_theorem.ML"*), which returns a prime number greater than the argument it receives. In file *"composition_with_lemmas.thy"* it can be found the result exposed in Section 4.4. We use the code extraction tool with a piece of code that we produced in the morphism based approach. The definition of morphisms as tuples already used in the morphism based approach (see Section 3.7) is given, as well as the composition of such tuples. Code is extracted from the definition of the composition of morphisms, showing the compatibility of the definition with the code extraction tool. The result can be found in file *"Composition.ML"*.

A.3 Detailed list of files

The files presented in this memoir can be found at the web page `http://www.unirioja.es/cu/jearansa/isabellefiles/`. They are presented in *html* format. In addition to this, and changing the *"html"* extension of the files by a *"thy"* extension in their URL, the Isabelle source files are also available. All files have been developed with an Intel Pentium M processor, 1.60 GHz. The operating system was MandrakeLinux 10.1 with the 2.6.8.1-12mdk kernel. The following programs were needed to compile the *"thy"* files:

- xemacs 21.1.14

- Isabelle2004
- Proof General 3.5
- Poly/ML 4.1.3

The files that can be found in the web page (in *html* format) are the following:

File Name	Number of lines
`symbolic/symbolic.html`	ca. 150
`set_theoretic/lemma1/lemma1_previous.html`	ca. 160
`set_theoretic/lemma1/lemma1_isom_sets.html`	ca. 250
`set_theoretic/lemma1/lemma1_isom_groups.html`	ca. 480
`set_theoretic/lemma1/lemma1_isom_ab_groups.html`	ca. 180
`set_theoretic/lemma1/lemma1_isom_cc.html`	ca. 700
`set_theoretic/lemma2/Kernel.html`	ca. 290
`set_theoretic/lemma2/Reduction.html`	ca. 540
`set_theoretic/lemma2/lemma2_is_reduct.html`	ca. 180
`triples/basic_def_2.html`	ca. 750
`triples/MRP_equiv.html`	ca. 140
`triples/group_mrp_ring.html`	ca. 1540
`inst_locales/HomGroup.html`	ca. 90
`inst_locales/HomGroupRestrict2.html`	ca. 520
`inst_locales/HomGroupComplection.html`	ca. 640
`inst_locales/HomGroupsComplection.html`	ca. 90
`inst_locales/lemma2_properties_1_to_4.html`	ca. 380
`ProgramExtraction/Primes/Monoid.html`	ca. 40
`ProgramExtraction/Primes/Monoid.ML.html`	ca. 25
`ProgramExtraction/Primes/Prime_number_theorem.html`	ca. 1470
`ProgramExtraction/Primes/Prime_number_theorem.ML.html`	ca. 60
`ProgramExtraction/Primes/composition_with_lemmas.html`	ca. 145
`ProgramExtraction/Primes/Composition.ML.html`	ca. 30

Table A.1: List of files

Bibliography

[Foc, 2005] (2005). *FoCaL Reference Manual.* The Foc Development team. http://focal.inria.fr/.

[Isa, 2005] (2005). *Isabelle Theory Library.* http://isabelle.in.tum.de/library/.

[Andrés et al., 2003] Andrés, M., García, F. J., Pascual, V., and Rubio, J. (2003). XML-Based interoperability among symbolic computation systems. In *ICWI 2003, IADIS International Conference WWW/Internet 2003, Algarve, Portugal, November 2003*, pages 925–928. IADIS Press.

[Aransay et al., 2002a] Aransay, J., Ballarin, C., and Rubio, J. (2002a). Deduction and Computation in Algebraic Topology. In *IDEIA 2002, IBERAMIA 2002, I Taller Iberoamericano sobre Deducción Automática e Inteligencia Artificial, Sevilla, Spain, October 2002*, pages 47–54. Universidad de Sevilla.

[Aransay et al., 2002b] Aransay, J., Ballarin, C., and Rubio, J. (2002b). Mechanizing proofs in Homological Algebra. In Zimmer, J. and Benzmüller, C., editors, *Calculemus Autumn School 2002: Poster Abstracts*, volume SR-02-06, pages 13–19. Universität des Saarlandes.

[Aransay et al., 2002c] Aransay, J., Ballarin, C., and Rubio, J. (2002c). Towards an automated proof of the Basic Perturbation Lemma. In Giménez, P., editor, *EACA 2002, Octavo Encuentro de Álgebra Computacional y Aplicaciones, Peñaranda de Duero, Spain, September 2002*, pages 91–95. Universidad de Valladolid.

[Aransay et al., 2003] Aransay, J., Ballarin, C., and Rubio, J. (2003). Towards a higher reasoning level in formalized Homological Algebra. In Hardin, T. and Rioboo, R., editors, *Calculemus 2003, 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Rome, Italy, September 2003*, pages 84–88. Aracne Editrice S.R.L.

[Aransay et al., 2004] Aransay, J., Ballarin, C., and Rubio, J. (2004). Four approaches to automated reasoning with differential algebraic structures. In Buchberger, B. and Campbell, J. A., editors, *AISC 2004, 7th International Conference on Artificial Intelligence and Symbolic Computation, Linz, Austria, September 2004*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 222–235. Springer.

[Aransay et al., 2005] Aransay, J., Ballarin, C., and Rubio, J. (2005). Extracting computer algebra programs from statements. In Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A., editors, *EUROCAST 2005, 10th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 2005*, volume 3643 of *Lecture Notes in Computer Science*, pages 159–168. Springer.

[Avigad, 2004] Avigad, J. (2004). notes on a formalization of the prime number theorem. Technical Report CMU-PHIL-163, Carnegie Mellon.

[Ballarin, 1999] Ballarin, C. (1999). *Computer Algebra and Theorem Proving.* PhD thesis, University of Cambridge.

[Ballarin, 2004] Ballarin, C. (2004). Locales and locale expressions in Isabelle/Isar. In Berardi, S., Coppo, M., and Damiani, F., editors, *TYPES 2003, 3rd International Workshop on Types for Proofs and Programs, Torino, Italy, May 2003*, volume 3085 of *Lecture Notes in Computer Science*, pages 34–50. Springer.

[Barnes and Lambe, 1991] Barnes, D. and Lambe, L. (1991). Fixed point approach to Homological Perturbation Theory. *Proceedings of the American Mathematical Society*, 112(3):881–892.

[Bauer and Wenzel, 2000] Bauer, G. and Wenzel, M. (2000). Computer-assisted mathematics at work (the Hahn-Banach Theorem in Isabelle/Isar). In Coquand, T., Dybjer, P., Nordström, B., and Smith, J., editors, *TYPES'99, Types for Proofs and Programs International Workshop, Lökeberg, Sweden, June 1999*, volume 1956 of *Lecture Notes in Computer Science*, pages 61–76. Springer.

[Bauer and Wenzel, 2001] Bauer, G. and Wenzel, M. (2001). Calculational reasoning revisited - an Isabelle/Isar experience. In Boulton, R. J. and Jackson, P. B., editors, *TPHOLs'2001, 14th International Conference on Theorem Proving in Higher Order Logics, Edinburgh, Scotland, September 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 75–91. Springer.

[Berghofer, 2003a] Berghofer, S. (2003a). Program extraction in simply-typed higher order logic. In Geuvers, H. and Wiedijk, F., editors, *TYPES 2002, 2nd International Workshop on Types for Proofs and Programs, Berg en Dal, The Netherlands, April 2002*, volume 2646 of *Lecture Notes in Computer Science*, pages 21–38. Springer.

[Berghofer, 2003b] Berghofer, S. (2003b). *Proofs, Programs and Executable Specifications in Higher Order Logic.* PhD thesis, Technische Universität München.

[Berghofer, 2004] Berghofer, S. (2004). A constructive proof of Highman's Lemma in Isabelle. In Berardi, S., Coppo, M., and Damiani, F., editors, *TYPES 2003, 3rd International Workshop on Types for Proofs and Programs, Torino, Italy, May 2003*, volume 3085 of *Lecture Notes in Computer Science*, pages 66–82. Springer.

[Berghofer, 2005] Berghofer, S. (2005). Answer to Tom Ridge. Available at the mail list isabelle-users@cl.cam.ac.uk, February 18, http://www.cl.cam.ac.uk/users/lcp/archive/.

[Bertot and Castéran, 2004] Bertot, Y. and Castéran, P. (2004). *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, volume 25 of *Texts in Theoretical Computer Science*. Springer.

[Boulmé et al., 1999] Boulmé, S., Hardin, T., Hirschkoff, D., Ménissier-Morain, V., and Rioboo, R. (1999). On the way to certify Computer Algebra systems. In *Proceedings of the Calculemus workshop of FLOC'99 (Federated Logic Conference, Trento, Italie)*, volume 23 of *Electronic Notes in Theoretical Computer Science*. Elsevier.

[Bourbaki, 1970] Bourbaki, N. (1970). *Éléments de mathématique. Algèbre. Chapitres 1 à 3*. Hermann.

[Brown, 1965] Brown, R. (1965). The twisted Eilenberg-Zilber theorem. In *Celebrazioni Archimedi de Secolo XX, Simposio di Topologia (Messina, 1964)*, pages 33–37. Edizione Oderisi.

[Calmet, 2003] Calmet, J. (2003). Some grand mathematical challenges in mechanized mathematics. In Hardin, T. and Rioboo, R., editors, *Calculemus 2003, 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Rome, Italy, September 2003*, pages 137–141. Aracne Editrice S.R.L.

[Church, 1940] Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68.

[Constable et al., 1986] Constable, R. L., Allen, S. F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, D. J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, J. T., and Smith, S. F. (1986). *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ.

[Coquand and Huet, 1988] Coquand, T. and Huet, G. (1988). The calculus of constructions. *Information and Computation*, 76(2,3):95–120.

[Cruz-Filipe and Spitters, 2003] Cruz-Filipe, L. and Spitters, B. (2003). Program extraction from large proof developments. In Basin, D. and Wolff, B., editors, *TPHOLs'2003, 16th International Conference on Theorem Proving in Higher Order Logics, Rome, Italy, September 2003*, volume 2758 of *Lecture Notes in Computer Science*, pages 205–220. Springer.

[Davenport, 1981] Davenport, J. (1981). Effective mathematics—the computer algebra viewpoint. In *Constructive Mathematics*, volume 873 of *Lecture Notes in Mathematics*, pages 31–43. Springer.

[Davenport, 1989] Davenport, J. (1989). Algebraic computations and structures. In *Computer Algebra*, volume 113 of *Lecture Notes in Pure and Applied Mathematics*, pages 129–144. Dekker.

[Davis, 2001] Davis, M. (2001). The early history of Automated Deduction. In Robinson, A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, volume I, chapter 1, pages 3–15. Elsevier Science.

[Dousson et al., 1999] Dousson, X., Sergeraert, F., and Siret, Y. (1999). The Kenzo program. http://www-fourier.ujf-grenoble.fr/~{}sergerar/Kenzo/.

[Dybjer et al., 2004] Dybjer, P., Haiyan, Q., and Takeyama, M. (2004). Verifying Haskell programs by combining testing, model checking and interactive theorem proving. *Information and Software Technology*, 46(15):1011–1025.

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

[Glimming, 2001] Glimming, J. (2001). Logic and automation for Algebra of programming. Master's thesis, Maths Institute, University of Oxford. http://www.nada.kth.se/%7eglimming/publications.shtml.

[Gordon and Melham, 1993] Gordon, M. J. C. and Melham, T. F., editors (1993). *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press.

[Gordon et al., 1979] Gordon, M. J. C., Milner, A. J., and Wadsworth, C. P. (1979). *Edinburgh LCF - A mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer.

[Graham, 1996] Graham, P. (1996). *ANSI Common Lisp*. Prentice Hall.

[Gugenheim, 1972] Gugenheim, V. K. A. M. (1972). On the chain complex of a fibration. *Illinois Journal of Mathematics*, 16(3):398–414.

[Harrison, 1996] Harrison, J. (1996). Formalized Mathematics. Technical Report 36, Turku Centre for Computer Science (TUCS). http://www.cl.cam.ac.uk/users/jrh/papers/form-math3.html.

[Hoare, 1972] Hoare, C. A. R. (1972). Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281.

[Jacobson, 1995] Jacobson, N. (1995). *Basic Algebra I*. W. H. Freeman and Company.

[Kammüller, 1999] Kammüller, F. (1999). Modular structures as dependent types in Isabelle. In Altenkirch, T., Naraschewski, W., and Reus, B., editors, *TYPES 98, International Workshop on Types for Proofs and Programs, Kloster Irsee, Germany, March 1998*, volume 1657 of *Lecture Notes in Computer Science*, pages 121–133. Springer.

[Kammüller and Paulson, 1999] Kammüller, F. and Paulson, L. C. (1999). A formal proof of Sylow's Theorem – an experiment in Abstract Algebra with Isabelle/HOL. *Journal of Automated Reasoning*, 23(3):235, 264.

[Kaufmann et al., 2000] Kaufmann, M., Manolios, P., and Strother-Moore, J. (2000). *Computer-Aided Reasoning: An Approach*. Kluwer Academic Press.

[Kopylov and Nogin, 2001] Kopylov, A. and Nogin, A. (2001). Markov's principle for propositional type theory. In Fribourg, L., editor, *CSL 2001, 15th International Workshop on Computer Science Logic. 10th Annual Conference of the EACSL, Paris, France, September 2001*, volume 2142 of *Lecture Notes in Computer Science*, pages 570–584. Springer.

[Lambán et al., 2003] Lambán, L., Pascual, V., and Rubio, J. (2003). An object-oriented interpretation of the EAT system. *Applicable Algebra in Engineering, Communication and Computation*, 14(3):187–215.

[Leroy et al., 2005] Leroy, X., Doligez, D., Garrigue, J., Rémy, D., and Vouillon, J. (2005). *The Objective Caml system release 3.09.* Institut National de Recherche en Informatique et en Automatique. http://caml.inria.fr/distrib/ocaml-3.09/ocaml-3.09-refman.pdf.

[Loeckx et al., 1996] Loeckx, J., Ehrich, H., and Wolf, M. (1996). *Specification of Abstrac Data Types.* Wiley and Teubner.

[Mac Lane, 1994] Mac Lane, S. (1994). *Homology.* Springer.

[Markov, 1971] Markov, A. A. (1971). On constructive mathematics. *American Mathematical Society Translations*, 2(98):1–9.

[May, 1967] May, J. P. (1967). *Simplicial objects in Algebraic Topology*, volume 11 of *Van Nostrand Mathematical Studies.* D. Van Nostrand Co.

[McCune, 2003] McCune, W. (2003). Otter 3.3 Reference Manual. Technical Memorandum 263, Argonne National Laboratory, Mathematics an Computer Science Division. http://www-unix.mcs.anl.gov/AR/otter/otter33.pdf.

[Müller and Slind, 1997] Müller, O. and Slind, K. (1997). Treating partiality in a logic of total functions. *The Computer Journal*, 40(10):640–652.

[Naraschewski and Wenzel, 1998] Naraschewski, W. and Wenzel, M. (1998). Object-oriented verification based on record subtyping in higher-order logic. In Grundy, J. and Newey, M., editors, *TPHOLs'98, 11th International Conference on Theorem Proving in Higher Order Logics, Canberra, Australia, September 1998*, volume 1479 of *Lecture Notes in Computer Science*, pages 349–366. Springer.

[Nipkow et al., 2000] Nipkow, T., Paulson, L. C., and Wenzel, M. (2000). *Isabelle's Logics: HOL.*

[Nipkow et al., 2002] Nipkow, T., Paulson, L. C., and Wenzel, M. (2002). *Isabelle/HOL: A proof assistant for higher order logic*, volume 2283 of *Lecture Notes in Computer Science.* Springer.

[O'Keefe, 2004] O'Keefe, G. (2004). Towards a readable formalisation of Category Theory. In Atkinson, M., editor, *Computing: The Australasian Theory Symposium*, volume 91 of *Electronic Notes in Theoretical Computer Science*, pages 212–228. Elsevier.

[Pascual, 2002] Pascual, V. (2002). *Objetos localmente efectivos y tipos abstractos de datos*. PhD thesis, Universidad de La Rioja.

[Paulson, 1989] Paulson, L. C. (1989). The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397.

[Paulson, 1990a] Paulson, L. C. (1990a). A formulation of the simple theory of types (for Isabelle). In Martin-Löf, P. and Mints, G., editors, *COLOG-88, International Conference on Computer Logic, Tallinn, USSR, December 1988*, volume 417 of *Lecture Notes in Computer Science*, pages 246–274. Springer.

[Paulson, 1990b] Paulson, L. C. (1990b). Isabelle: The next 700 theorem provers. In Odifreddi, P., editor, *Logic and Computer Science*, pages 361–386. Academic Press.

[Paulson, 1992] Paulson, L. C. (1992). Designing a theorem prover. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science*, volume 2, pages 415–475. Oxford University Press.

[Paulson, 1994] Paulson, L. C. (1994). *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer.

[Paulson, 1996] Paulson, L. C. (1996). *ML for the working programmer*. Cambridge University Press, 2nd edition.

[Paulson, 2004] Paulson, L. C. (2004). Defining functions on equivalence classes. http://www.cl.cam.ac.uk/users/lcp/papers/Reports/equivclasses.pdf.

[Rubio, 2004] Rubio, J. (2004). Emulating proof-by-hand in Isabelle. In *Mathematics, Algorithms and Proofs*. http://www.disi.unige.it/map/luminy/slides/rubio.ppt.

[Rubio, 2005] Rubio, J. (2005). Constructive proofs or constructive statements? In *Dagstuhl Proceedings*, volume 050201. http://www.dagstuhl.de/05021/.

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

[Rubio and Sergeraert, 1997] Rubio, J. and Sergeraert, F. (1997). Constructive Algebraic Topology. Lecture Notes Summer School in Fundamental Algebraic Topology, Institut Fourier. http://www-fourier.ujf-grenoble.fr/~{}sergerar/Summer-School/.

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

[Rubio et al., 1997] Rubio, J., Sergeraert, F., and Siret, Y. (1997). EAT: Symbolic software for effective homology computation. Technical report, Institut Fourier, Grenoble. ftp://fourier.ujf-grenoble.fr/pub/EAT.

[Rubio et al., 1998] Rubio, J., Sergeraert, F., and Siret, Y. (1998). Overview of EAT, a system for effective homology computation. *The Symbolic and Algebraic Computation Newsletter*, 3:69–79.

[Schirmer, 2005] Schirmer, N. (2005). A verification environment for sequential imperative programs in Isabelle/HOL. In Baader, F. and Voronkov, A., editors, *LPAR 2004, 11th International Workshop on Logic for Programming, Artificial Intelligence, and Reasoning, Montevideo, Uruguay, March 2005*, volume 3452 of *Lecture Notes in Artificial Inteligence*, pages 398–414. Springer.

[Shih, 1962] Shih, W. (1962). Homologie des espaces fibrés. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 13:1–88.

[Steele, 1990] Steele, G. L. (1990). *Common Lisp, the Language*. Digital Press.

[Wenzel, 2002] Wenzel, M. (2002). *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Technische Universität München.

[Wenzel, 2004] Wenzel, M. (2004). The Isabelle/Isar reference manual. Technical report, Technische Universität München. http://isabelle.in.tum.de/dist/Isabelle/doc/isar-ref.pdf.

[Wenzel, 2005] Wenzel, M. (2005). Using axiomatic type classes in Isabelle. Technical report, Technische Universität München. http://isabelle.in.tum.de/dist/Isabelle/doc/axclass.pdf.

[Wenzel and Wiedijk, 2002] Wenzel, M. and Wiedijk, F. (2002). A comparison of Mizar and Isar. *Journal of Automated Reasoning*, 29(3,4):389–411.

[Wiedijk, 2003] Wiedijk, F. (2003). Comparing mathematical provers. In Asperti, A., Buchberger, B., and Davenport, J., editors, *MKM 2003, 2nd International Conference on Mathematical Knowledge Management, Bertinoro, Italy, February 2003*, volume 2594 of *Lecture Notes in Computer Science*, pages 188–202. Springer.

Razonamiento mecanizado en Álgebra Homológica

Memoria presentada para la obtención
del título de Doctor

Jesús María Aransay Azofra

Directores: Dr. D. Julio Rubio García
Dr. D. Clemens Ballarin

Universidad de La Rioja

Departamento de Matemáticas y Computación

Logroño, Enero de 2006

Este trabajo ha sido parcialmente subvencionado por los proyectos DGES (PB98-1621-C02) y SEUI-MEC (TIC2002-01626) del Gobierno Español, el projecto CALCULEMUS (HPRN-CT-2000-00102) de la Unión Europea, el proyecto ACPI2002/06 de la Comunidad de La Rioja, y el proyecto API-02/24 y las becas ATUR02/03, ATUR03/03, ATUR04/04 de la Universidad de La Rioja.

Agradecimientos

Julio Rubio y Clemens Ballarin hicieron posible este trabajo. Julio, desde la cercanía, me concedió la oportunidad de iniciarme en el camino de la investigación. Gracias a su interés disfruté de excelentes oportunidades académicas. Sin embargo, lo que más he de agradecerle es su disponibilidad para tratar todas las ideas, cuestiones o simples comentarios que fueron surgiendo a lo largo de la elaboración de la tesis, así como su cordialidad. Su entusiasmo me ayudó a superar muchos momentos difíciles. Clemens y su atención supusieron una gran motivación para seguir adelante. A pesar de la distancia que nos separó durante la elaboración de la mayor parte de esta memoria, siempre se mostró accesible y dispuesto a colaborar en todo lo necesario.

Dentro de nuestro grupo de investigación en la Universidad de La Rioja, debo agradecer la labor de Laureano Lambán. Sus palabras siempre fueron de ánimo y sus consejos útiles. Vico Pascual y César Domínguez fueron un gran ejemplo a seguir e intentaron apoyarme con su experiencia. Con Juan José Olarte compartí mis primeros trabajos de investigación. Ana Romero, Mirian Andrés, Ángel Luis Rubio, Francisco José García y Eloy Mata estuvieron dispuestos a escuchar y valorar cada una de las ideas que forman parte de este trabajo. Aprovecho también la ocasión para agradecer a los integrantes del Departamento de Matemáticas y Computación de la Universidad de La Rioja el trato que siempre me han deparado. No sería justo olvidar a todos aquellos amigos y compañeros que alguna vez pasaron por el despacho, aunque fuera con la simple intención de tomar un café o charlar un rato.

Jacques Calmet me concedió la oportunidad de trabajar durante seis meses en el Institüt für Algorithmen und Kognitive Systeme de la Universität Karlsruhe. Esa estancia sirvió para definir los problemas que se abordan en esta memoria, y a nivel personal, para comprobar lo difícil y satisfactoria que puede llegar a ser la investigación.

El grupo de Tobias Nipkow en la Technische Universität München me acogió de forma excelente durante diez semanas. Trabajar y convivir con todos ellos fue una gran experiencia.

Finalmente, he de agradecer a mi familia el haber creído en mí y en lo que hacía. Mis padres y mi hermano Óscar siempre me apoyaron y confiaron en mí. Judith, con su ánimo, su paciencia y su cariño, hizo que todo esto fuera más fácil y gratificante.

A todos ellos, gracias.

Índice General

Introducción

Las Matemáticas no son una ciencia formal. O, al menos, no son una ciencia formal en el sentido estricto de la lógica simbólica. Los matemáticos, generalmente, tienen mucha más libertad con respecto a los aspectos fundacionales, o al menos, formales. Por ejemplo, cuando se aplica un resultado previo, se presta poca atención a demostrar que todas las condiciones previas se satisfacen, excepto en los casos en los que una condición previa parece difícil de ser probada o interesante por sí misma. Obviamente, nociones tales como las de *dificultad* o *interés* son bastante elusivas desde un punto de vista formal.

Sin embargo, éste no es un tema completamente subjetivo. El nivel de detalle parece ser una cuestión social que depende de múltiples variables, tales como la audiencia a la que va destinado el texto (menor detalle en notas de investigación, más detalle en libros de texto) o la disciplina en sí misma (por ejemplo, es bien sabido que algunos argumentos en física teórica no son considerados suficientemente rigurosos desde un punto de vista estrictamente matemático).

Con respecto a la lógica en la que se basa el razonamiento, todavía se suele ser más difuso. Respecto a los fundamentos, generalmente se suele confiar en algún tipo de teoría intuitiva de conjuntos, implícitamente basada, de una u otra forma, en el sistema axiomático de Zermelo-Fraenkel.

Las *variables* son otra fuente de inexactitud en los textos matemáticos. Muy a menudo, el tipo de las variables (es decir, el rango de valores por los que una variable puede ser reemplazada) es indeterminado o basado en un convenio tipográfico. Las variables no siempre están claramente definidas como libres o ligadas, y en este segundo caso, es bastante inusual ser completamente preciso en temas como el rango de la cuantificación o el alcance de la ligadura. En este contexto, es bastante frecuente renombrar una variable en algunos sitios, pero no en todas las ocurrencias apropiadas.

Dos consecuencias de esta presentación informal de las Matemáticas, de muy diferente naturaleza (e importancia), pueden ser destacadas. Por una parte, una cierta cantidad de demostraciones matemáticas publicadas no pueden ser consideradas como tales, ya que contienen errores (generalmente muy menores) desde un punto de vista formal. Por otra parte, cierto grado de distanciamiento de las (tediosas) exigencias de la lógica simbólica permite el desarrollo de las matemáticas de una forma extremadamente rápida y potente, que puede ser considerada como una característica del área en general.

La razón para este comportamiento de las matemáticas es que los matemáticos han alcanzado un alto nivel de razonamiento, y, aún más, parecen ser capaces de incrementar de forma ilimitada el nivel de abstracción de su razonamiento.

¿Por qué puede convivir cierto grado de inexactitud con la firme evolución de las matemáticas? La respuesta depende de las diferencias entre tres conceptos relacionados:

- la *idea* sobre la que se basa una demostración;
- la presentación de la demostración en un texto matemático;
- la demostración en el sentido de la lógica simbólica.

Si una idea es correcta, entonces se acepta que su presentación puede ser hecha en una forma (más o menos) imprecisa. Por supuesto, la idea es *correcta* si puede ser convertida (de un modo más o menos elaborado) en una demostración en el sentido formal.

En la interrelación entre estas tres entidades es donde se nos muestra la relevancia de los demostradores de teoremas. Un demostrador de teoremas es un programa de ordenador diseñado con el propósito de construir demostraciones en el sentido lógico. Desde un punto de vista utópico, un demostrador de teoremas debería ser capaz de, dada la idea de una demostración, obtener la correspondiente demostración formal de una forma automática.

De hecho, históricamente, la meta de los primeros demostradores de teoremas era incluso más pretenciosa: dada una afirmación, el sistema debería ser capaz, automáticamente, de demostrarla o refutarla, y en el primer supuesto, debería encontrar una prueba formal de la afirmación (que entonces pasaría a ser considerada un teorema). Estas expectativas se vieron alimentadas por la solución completa del caso proposicional, y también por el algoritmo de resolución de Robinson. De aquí surgió la primera generación de demostradores automáticos de teoremas (ver [Davis, 2001] para una revisión histórica de estos primeros sistemas).

Sin embargo, y a pesar del éxito de sistemas tales como OTTER [McCune, 2003], prontamente se reconoció que una cierta intervención humana sería necesaria, tanto para definir las afirmaciones (en la formulación precisa del demostrador automático de teoremas[1]) como para ofrecer al sistema guías que le ayuden a dirigir las demostraciones. En los sistemas más automatizados, como pueden ser OTTER o ACL2 (para ACL2 se puede consultar, por ejemplo [Kaufmann et al., 2000]), la intervención mencionada se organiza en forma de *lemas auxiliares*.

Las posteriores generaciones de demostradores de teoremas surgieron del reconocimiento de la necesidad de esta intervención humana: son los *demostradores tácticos de teoremas*. Estos programas permiten dirigir interactivamente la construcción de una prueba formal. Los más relevantes en nuestro contexto son Isabelle y Coq. Isabelle es un asistente para la demostración en lógica de orden superior [Nipkow et al., 2002], diseñado

[1]Esta tarea, en los casos no triviales, es significativamente importante.

como un demostrador de teoremas que es *genérico* con respecto a las lógicas que implementa. Coq es un demostrador interactivo de teoremas [Bertot and Castéran, 2004], basado en el *cálculo inductivo de construcciones* [Coquand and Huet, 1988].

Hoy en día, cada demostrador de teoremas se puede interpretar como un punto en la línea que va desde la demostración automática de teoremas (donde la entrada es una afirmación) a los verificadores de demostraciones (donde la entrada es una supuesta demostración formal). Como puntos intermedios se pueden encontrar sistemas que permiten al usuario dirigir la construcción de la prueba, y, en particular, los demostradores tácticos de teoremas. Pero es preciso notar que las sugerencias que pueden ser dadas a los demostradores de teoremas existentes están mucho más cercanas a la lógica simbólica que a argumentos matemáticos de alto nivel (como pueden ser encontrados en cualquier texto matemático). Incluso si algunos teoremas interesantes han podido ser demostrados con estas herramientas (por ejemplo, el *Teorema Fundamental del Álgebra* demostrado en Coq [Geuvers et al., 2002]), el tamaño y la complejidad de estos trabajos ilustra adecuadamente el estado actual de los demostradores de teoremas, con respecto al legado matemático heredado durante siglos.

Consecuentemente, se hace necesario una mayor investigación con el objetivo de hacer realmente útiles los demostradores de teoremas a la comunidad matemática. Esta memoria se centra en uno de los aspectos de dicha investigación. De hecho, nuestra motivación inicial no era de una naturaleza genérica o filosófica. Nuestro interés se centra en un emergente campo de aplicación de los demostradores de teoremas: demostrar la corrección de programas de ordenador.

Más concretamente, el primer objeto de interés es un sistema de Álgebra Computacional para el cálculo en Topología Algebraica llamado *Kenzo*. Los sistemas de software EAT [Rubio et al., 1997] y Kenzo [Dousson et al., 1999] fueron escritos bajo la dirección de Sergeraert para el Cálculo Simbólico en Topología Algebraica y Álgebra Homológica. Estos sistemas han producido resultados relevantes (por ejemplo, algunos grupos de homología de espacios de lazos iterados) previamente desconocidos. Ambos sistemas se caracterizan por un uso intensivo de algunas técnicas de programación funcional, que en particular permiten codificar y manipular en tiempo de ejecución las estructuras de datos infinitas que aparecen en los algoritmos de Topología Algebraica. Como fue señalado en [Calmet, 2003], la Topología Algebraica es una de las áreas donde todavía se pueden encontrar problemas que supongan un reto para los sistemas de Álgebra Computacional y los demostradores de teoremas.

Con el objetivo de incrementar el conocimiento sobre estos sistemas, se puso en marcha hace algunos años un proyecto para analizar formalmente fragmentos de estos programas. En los últimos años, algunos resultados relativos a la especificación algebraica de las estructuras de datos han sido encontrados; véase, por ejemplo, [Lambán et al., 2003]. Continuando con este proceso, los algoritmos en los que aparecen estas estructuras de datos son ahora nuestro objetivo, con el propósito de obtener versiones certificadas de algunos fragmentos cruciales de Kenzo, usando el demostrador táctico de teoremas Isabelle. Los trabajos previos en Teoría de Grupos y la expresividad de la lógica de orden superior fueron las razones que nos llevaron a escoger Isabelle para nuestro trabajo.

Un primer resultado para el cual tratamos de implementar una demostración es el Lema Básico de Perturbación, ya que su demostración tiene un algoritmo asociado que es usado en Kenzo como una de las partes centrales del programa.

Sin embargo, en esta memoria no se presenta una demostración mecanizada completa del Lema Básico de Perturbación. A lo largo del proceso de investigación, nuestro objetivo varió hacia el estudio de técnicas que doten a los sistemas con un mayor nivel de razonamiento. Por lo tanto, preferimos explorar diferentes aproximaciones en un (pequeño) fragmento de la demostración, en lugar de incrementar el número de fragmentos demostrados.

Nuestra perspectiva es la de un usuario final. Es decir, no se ha considerado la posibilidad de introducir cambios en Isabelle (aún cuando nuestro trabajo pueda inspirar algunas nuevas características en las versiones posteriores de Isabelle, en particular en el mecanismo de interpretación de *locales*). Tampoco nuestro punto de vista es el del ingeniero de sistemas, sino el del matemático computacional. Por lo tanto, hemos perseguido la introducción de esquemas conceptuales, con el propósito de incrementar el nivel de razonamiento del sistema (Isabelle), *en su estado actual*. Consideramos que algunos de esos esquemas han sido encontrados, como pretendemos ilustrar en esta memoria.

La organización de la memoria es la siguiente. En el Capítulo 1, introducimos algunos preliminares: en Matemáticas (Álgebra Homológica elemental), demostradores de teoremas (Isabelle), y Álgebra Computacional. El Capítulo 2 está dedicado a una demostración extremadamente detallada del Lema de Perturbación Básico. El nivel de detalle es mucho más fino que en un libro de texto convencional. Esta (un poco) pesada presentación se justifica por tres motivos. El primero es hacer consciente al lector (y previamente, al autor) del gran número de detalles que generalmente pasan inadvertidos en cualquier argumento matemático simple. El segundo motivo es servir como diseño (de nivel medio) de una demostración mecanizada. El tercero es dejar claro que, incluso con esta tediosa presentación, la distancia con un demostrador de teoremas es muy grande. Este hecho quedará claro cuando se lea el Capítulo 3, donde comienza la codificación en Isabelle de (fragmentos de) la demostración.

El Capítulo 3 contiene nuestras contribuciones principales. Después de introducir un lema que sirve de ejemplo, presentamos cuatro aproximaciones para mecanizar el razonamiento en Álgebra Homológica. La primera aproximación es llamada *simbólica* y está caracterizada por su gran nivel de automatización y su baja capacidad expresiva. La segunda aproximación, llamada *conjuntista*, intenta emular la forma “teórica” de razonamiento habitual en Matemáticas (“teórica” porque las demostraciones se vuelven impracticables rápidamente debido a su gran tamaño). La tercera, llamada *basada en morfismos*, intenta usar las ventajas de las dos aproximaciones previas: automatización y poder expresivo. En la cuarta aproximación, se utiliza una mejora técnica recientemente incorporada a Isabelle (llamada *locales*), lo que permite hacer nuestras demostraciones más cortas y legibles.

El Capítulo 4 explora brevemente un sistema para unir demostraciones y programas. No pretendemos obtener programas más generales o más eficientes (esa aspiración queda fuera del alcance de la tecnología actual, como explicaremos más adelante). El objetivo general es más bien acercarnos a la meta inicial de nuestra investigación: dar certificados de corrección para para programas de Álgebra Computacional. Es interesante resaltar que desde este pequeño experimento hemos llegado, de forma un tanto inesperada, a un problema de fundamentos importante: a saber, el carácter constructivo de nuestro enfoque y, más en general, de la Topología Algebraica en sí misma.

La aproximación escogida es la *extracción de código*. De un texto formal (una demostración, una especificación) se extrae (automáticamente) un programa ejecutable. Este programa es, por construcción, correcto con respecto al texto formal usado como fuente. En el contexto de los demostradores de teoremas, la aproximación más usual es extraer código de las *demostraciones* (ver, por ejemplo, [Cruz-Filipe and Spitters, 2003]). Sin embargo, los enunciados en Topología Algebraica contienen, en la mayoría de los casos, la definición de los objetos que deben ser computados, es decir: son *enunciados constructivos*. Por lo tanto, el código puede ser extraído a partir de *los enunciados* en lugar de a partir de las demostraciones. Estas ideas serán ilustradas en el Capítulo 4 por medio de un ejemplo aritmético elemental (más concretamente, relacionado con los números primos). Al final del capítulo, se presenta un ejemplo simple de extracción de código en el área del Álgebra Computacional.

La memoria termina con un capítulo de conclusiones y problemas abiertos y la bibliografía. En el Apéndice A se puede encontrar una enumeración detallada de los ficheros de Isabelle que han sido desarrollados para este trabajo. Los ficheros de código Isabelle están disponibles en la página web `http://www.unirioja.es/cu/jearansa/isabellefiles/`. El número total de líneas de código Isabelle y ML es aproximadamente 8850, y el número de KB es superior a los 500.

Resumen de los capítulos

Preliminares

Introducimos las nociones básicas que van a ser utilizadas a lo largo de la memoria. Está dividido en tres secciones. La Sección 1.1 está dedicada a definiciones elementales en Álgebra Homológica. En la Sección 1.2 se introduce el sistema de Álgebra Computacional Kenzo. Finalmente en la Sección 1.3 se introduce el demostrador de teoremas Isabelle.

El Lema Básico de Perturbación

En el Capítulo 2 se presenta una demostración completa de un resultado relevante en el área del Algebra Homológica, generalmente conocido como Lema Básico de Perturbación. En la Sección 2.1 se destaca la importancia del Lema Básico de Perturbación, poniendo especial énfasis en sus aplicaciones. En la Sección 2.2 se detalla la demostración de este resultado. Finalmente, en la Sección 2.3 se enuncia una versión no graduada del mismo lema, que será la que luego abordaremos con ayuda de un demostrador mecanizado.

Mecanizando la demostración: estudio de un caso en Isabelle

En el Capítulo 3 se muestran los resultados hallados al tratar de obtener una demostración del Lema Básico de Perturbación en el demostrador de teoremas Isabelle. En la Sección 3.2 hacemos una exposición de los motivos que nos llevaron a escoger Isabelle entre los demostradores de teoremas disponibles. En la Sección 3.3 se describen, por medio de un ejemplo introductorio, las diferencias entre las cuatro aproximaciones que planteamos para resolver el problema. La Sección 3.4 presenta algunas herramientas de especificación algebraica que nos permitirán una descripción más precisa de la diferente representación del problema que se hace en cada una de las aproximaciones. En las Secciones 3.5, 3.6, 3.7 y 3.8 se introducen las cuatro aproximaciones, mediante la descripción de sus aspectos formales, la presentación de los fragmentos del Lema Básico de Perturbación que se han demostrado con cada una de ellas, y la comparación entre ellas.

Extracción de programas de Álgebra Computacional a partir de enunciados

En el Capítulo 4 se presenta una metodología que permite la obtención de programas certificados, así como varios ejemplos de uso. La metodología se basa en la extracción de código, mediante una herramienta dispuesta para ello en Isabelle. El código es extraído de definiciones, en lugar de extraerlo desde demostraciones, lo cual permite que la lógica empleada en esas demostraciones pueda ser más rica. Las definiciones corresponden a enunciados de teoremas que son, en sí mismos, "constructivos". Una vez esos teoremas han sido demostrados de forma mecanizada, podemos extraer código de las definiciones asociadas a ellos. En la Sección 4.2 introducimos el caso de estudio escogido en Kenzo: la composición de homomorfismos. Se presenta un fragmento de CLOS donde se muestra la definición de la composición. En la Sección 4.3 escogemos un dominio más conocido, la aritmética elemental, para resolver un ejemplo relacionado con la demostración de Euclides de la existencia de infinitos primos. El propósito es introducir las ideas principales sobre extracción de código evitando la complejidad del Álgebra Homológica. Las ideas introducidas nos permitirán extraer de nuestras definiciones en Isabelle un programa certificado ML que calcula un primo mayor que el natural que es su entrada. En la Sección 4.4 retomamos nuestro problema original sobre la composición de morfismos, y aplicando la misma metodología obtenemos un programa ML (certificado correcto por Isabelle) que compone dos morfismos.

Conclusiones y trabajo futuro

El Capítulo 5 está dividido en dos secciones. En la Sección 5.1 se presentan las conclusiones de nuestro trabajo. La Sección 5.2 presenta los problemas abiertos o aún sin concluir que quedan a raíz de nuestro trabajo.

Conclusiones y Trabajo Futuro

Conclusiones

Este trabajo debería ser entendido como un primer paso hacia la mecanización del razonamiento en Topología Algebraica, con énfasis en su aplicación a la demostración de la corrección de programas de Álgebra Computacional. Más concretamente, nuestra inspiración proviene del sistema Kenzo [Dousson et al., 1999], y de trabajos previos realizados para especificar dicho sistema [Lambán et al., 2003].

Nos hemos centrado en un resultado concreto, llamado Lema Básico de Perturbación. Después de explicar brevemente la relevancia del Lema Básico de Perturbación en el área de la Topología Algebraica algorítmica, se ha presentado una demostración muy detallada del mismo. Esta demostración sigue las directrices de una demostración originalmente presentada en [Rubio and Sergeraert, 1997]. Es interesante hacer notar que en [Rubio and Sergeraert, 1997] se emplearon unas 70 líneas de texto para describir la demostración (incluyendo el enunciado y demostración de un lema auxiliar), mientras que en nuestra presentación se han necesitado unas 20 páginas. Esta observación debería ser examinada a la luz de tres elementos mencionados en la introducción:

- la *idea* sobre la que se basa una demostración;
- la presentación de la demostración en un texto matemático;
- la demostración en el sentido de la lógica simbólica.

Nuestra demostración del Lema Básico de Perturbación muestra cómo una misma idea (punto 1) admite muy diferentes presentaciones (punto 2). El Capítulo 3 de la memoria ilustra claramente cómo, incluso en los casos de demostraciones exageradamente detalladas, una demostración completa en el sentido de la lógica simbólica (punto 3) continúa siendo excesivamente distante.

Dos aspectos principales han sido detectados en nuestra demostración (comparándola con la demostración en [Rubio and Sergeraert, 1997]) que motivan el tamaño de la presentación de la demostración. Por una parte, el hecho de que, cuando se razona con complejos de cadenas, se deben considerar gran variedad de morfismos, dependiendo de cómo respetan los grados y si son coherentes o no con respecto a las diferenciales. En casos extremos, una misma aplicación puede ser considerada como un morfismo de grupos o como un morfismo de complejos de cadenas en un mismo fragmento de demostración. Por otra parte, es bastante común restringir un morfismo a algún subgrupo (o a algún subcomplejo de cadenas) de su origen, y referirse al morfismo inicial y a su restricción de forma indistinta.

Ambos aspectos pueden ser considerados casos particulares de otro tema destacado en la introducción: la gran libertad para nombrar entidades en textos matemáticos estándar. Una misma variable (incluso una misma letra) puede denotar diferentes objetos (diferentes pero íntimamente relacionados en un sentido conceptual, por supuesto) en un mismo fragmento de demostración.

Esta cuestión de las variables parece difícil de ser abordada de forma global (es tan general que, en estos términos, se trata más bien de un problema mal planteado). No obstante, los dos casos particulares mencionados con anterioridad han sido tratados en nuestro trabajo. El primero en, se podría decir, una forma *interna* en Isabelle; el segundo en un modo que podría ser considerado *externo* a Isabelle.

El primer aspecto es solventado, en parte, usando una herramienta general de Isabelle/HOL: los *registros extensibles*. Este tipo de datos provee a Isabelle con cierto tipo de *herencia*. Por lo tanto, un complejo de cadenas puede ser también considerado simplemente como un grupo graduado por el sistema de tipos, y entonces los morfismos entre estas estructuras pueden ser más fácilmente tratados.

Para el segundo problema, el de la restricción, nuestra solución ha sido representar un morfismo como una estructura de datos más compleja. Es decir, hemos cambiado la forma en que son codificados los morfismos. Ésta es la razón por la que consideramos esta propuesta como *externa* a Isabelle: no depende de ningún recurso de Isabelle, sino de un nuevo esquema conceptual. Los morfismos son representados como ternas: además de la información funcional (que incluye el dominio y codominio del morfismo), el dominio y el codominio verdadero son también incorporados. Por lo tanto, una misma aplicación puede ser usada en diferentes morfismos, emulando la forma de trabajar de los matemáticos con respecto a las restricciones (y extensiones) de funciones.

Esta técnica de cambiar nuestra *perspectiva* de un objeto matemático dado (en el ejemplo mostrado: variando de un morfismo representado como una aplicación a un morfismo como una terna), es un punto central en el Capítulo 3, donde realmente empieza la codificación de la demostración del Lema Básico de Perturbación en Isabelle. Hemos desarrollado 4 aproximaciones diferentes para tratar de resolver los problemas para un pequeño fragmento de la demostración. La noción de *abstracción*, introducida por Hoare para estudiar la implementación de estructuras de datos [Hoare, 1972], nos permite analizar las diferentes aproximaciones, y, aún más, dirigir nuestra investigación en algunos puntos delicados.

Aunque la discusión realizada se restringe a un pequeño fragmento de la demostración, algunos de los problemas encontrados (y resueltos) son de naturaleza bastante general, ya que están relacionados con el razonamiento genérico con estructuras algebraicas y morfismos. (Otra simplificación mayor en el Capítulo 3 es la adopción de la versión *no graduada* del Lema Básico de Perturbación; no obstante consideramos esta simplificación como una cuestión técnica en un intento de separar problemas: los problemas de trabajar con estructuras graduadas son ortogonales a la cuestión central de implementar una demostración del Lema Básico de Perturbación en un demostrador de teoremas.)

Para entender las diferentes aproximaciones que presentamos es necesario saber que la demostración del Lema Básico de Perturbación utiliza morfismos. Entonces, un morfismo puede ser visto de dos formas:

- como elemento de una estructura algebraica (el anillo de endomorfismos, por ejemplo);
- como una aplicación entre conjuntos (satisfaciendo ciertas propiedades).

Es claro que la primera forma es incompleta, ya que nuestro interés está relacionado con el tratamiento algorítmico de morfismos (desde un punto de vista técnico, se puede decir que esta representación carece de *contenido computacional*). Pero también es cierto que la primera perspectiva permite el razonamiento con morfismos en un nivel de abstracción próximo al típico en un texto matemático estándar. Nuestras tres primeras aproximaciones exploran esta distinción (la cuarta es básicamente una reformulación de la tercera, usando una herramienta técnica de Isabelle, llamada *locales*).

En la primera aproximación, que denominamos *aproximación simbólica*, sólo la primera interpretación de los morfismos es considerada: un morfismo es representado por medio de un elemento genérico (un *símbolo*) de un anillo genérico. Si una propiedad relevante para el Lema Básico de Perturbación puede ser expresada en este contexto, se trata de un nivel de razonamiento muy eficiente. A este modo de razonamiento lo hemos denominado *razonamiento ecuacional*: las demostraciones son desarrolladas por reescritura de expresiones, aplicando los axiomas de estructuras algebraicas genéricas (como anillos o grupos). Esto permite un alto grado de automatización, usando la táctica de Isabelle *algebra* (desarrollada por C. Ballarin). Sin embargo, la expresividad de esta aproximación es baja (hay fragmentos de la demostración del Lema Básico de Perturbación que necesitan considerar los morfismos como aplicaciones; por ejemplo, cuando es necesario restringir un morfismo a un subconjunto distinguido de su dominio) y, como hemos mencionado previamente, carece de contenido computacional.

Por lo tanto, se necesitan nuevas ideas. En nuestra segunda aproximación consideramos los morfismos en la segunda forma citada anteriormente: una aplicación en el sentido de la teoría de conjuntos satisfaciendo ciertas propiedades. Esta aproximación ha sido nombrada *aproximación basada en teoría de conjuntos* o simplemente *aproximación conjuntista*. Este punto de vista es el más ortodoxo en un sentido simplista: desde la teoría de conjuntos, una jerarquía de estructuras algebraicas es construida, como en Álgebra Universal. Hemos intentado emular esta forma de trabajar en Isabelle, empezando desde las estructuras más elementales (tales como magma, semigrupo, monoide, ...). La principal herramienta para este fin han sido los *registros extensibles*, que dotan a Isabelle de características interesantes relacionadas con la herencia. Esto permite también tratar con los distintos tipos de morfismos que pueden ser definidos sobre una misma estructura algebraica, como mencionamos anteriormente.

A pesar de su validez, y de la ayuda técnica de Isabelle, rápidamente se hace evidente que las demostraciones no son factibles: descender siempre al razonamiento con los *elementos* de las estructuras algebraicas (en lugar de trabajar con las estructuras mismas, y con sus propiedades como tales) incrementa enormemente la longitud de las demostraciones, e, incluso peor, oculta los pasos lógicos importantes entre cantidad de tediosos detalles técnicos. La conclusión es la confirmación de algo ya conocido: los matemáticos hacen uso de las herramientas de bajo nivel en teoría de conjuntos sólo para establecer fundamentos; después, trabajan con un alto nivel de razonamiento.

La tercera aproximación intenta reunir las ventajas de las dos aproximaciones previas, evitando sus inconvenientes. En otras palabras, hemos intentado recobrar el razonamiento ecuacional (y por consiguiente, un alto nivel de automatización) sin perder expresividad y capacidad de cálculo. Para este propósito, ha sido necesario cambiar en profundidad la representación de los morfismos, tanto desde un punto de vista operacional como desde un punto de vista estructural.

Desde la primera perspectiva, ha sido necesario tratar la parcialidad, para poder conseguir una representación donde la *igualdad* en el modelo abstracto (es decir, la igualdad en matemáticas) provenga de la *igualdad* en la memoria del ordenador (en otras palabras, de la igualdad como objetos de Isabelle). Isabelle sólo admite funciones *totales* y los morfismos en Álgebra Homológica son también aplicaciones totales. Entonces, ¿dónde aparece la parcialidad? Las funciones parciales aparecen debido a la siguiente razón: las funciones Isabelle son totales con respecto a sus *tipos*. Pero los morfismos no son definidos sobre los tipos, sino sobre *conjuntos* que actúan como *soporte* (*carrier*, en inglés) de las estructuras algebraicas. Por lo tanto, cuando representamos un morfismo en Isabelle, sólo los datos del tipo al que pertenece el soporte tienen una imagen bien definida. Cada dato fuera del soporte *debe* tener una imagen (ya que las funciones en Isabelle son totales) pero las Matemáticas no dicen nada sobre la imagen, que es definida *arbitrariamente*. Por lo tanto, cuando queremos utilizar la igualdad de Isabelle entre dos morfismos (y esto es necesario si el *razonamiento ecuacional* ha de funcionar), resulta que el sistema no es capaz de decidir, debido a los valores arbitrarios de la función Isabelle fuera del soporte. Nuestra solución consiste en enviar cada dato fuera del soporte al elemento neutro de la estructura algebraica que representa el codominio de la función Isabelle. Esta *compleción* implica que dos morfismos iguales en el modelo abstracto son representados en el ordenador por dos objetos que son reconocidos como iguales por Isabelle (en términos técnicos, la representación es *fiel*: la función de abstracción es inyectiva). Entonces, el razonamiento ecuacional es, en principio, posible.

Pero, desde un punto de vista estructural, encontramos el problema de las variables mencionado anteriormente: la demostración del Lema Básico de Perturbación usa intensivamente que la misma aplicación puede definir diferentes morfismos, cambiando su dominio o codominio. Estos elementos son generalmente denotados por la misma variable, permitiendo al usuario demostraciones más cortas y claras. Para obtener este nivel de abstracción más alto en Isabelle, cambiamos una vez más nuestra representación de los morfismos. Ahora, una (compleción de) función es codificada con un dominio y un codominio. Estas ternas nos permiten pasar de un morfismo a una de sus restricciones (por ejemplo), sin reconstruir la estructura completa: sólo se requiere la modificación de uno de sus campos.

Una vez que estas modificaciones operacionales y estructurales han sido dispuestas, es necesario demostrar que el conjunto de los endomorfismos de un grupo conmutativo (representados como ternas) en Isabelle puede ser dotado de una estructura de anillo con las operaciones correspondientes. Como era previsible, el número de líneas de código para demostrar este teorema en Isabelle es bastante considerable, siendo mayor que en las demostraciones desarrolladas en la segunda aproximación (¡que hemos calificado como *no factibles*!). Esto es bastante natural: el trabajo necesario para una *infraestructura* puede ser asumido, debido a la importancia de la reutilización que puede ser hecha de esos resultados. En nuestro caso, en particular, el *razonamiento ecuacional* es completamente recuperado con esta aproximación, incluida la automatización obtenida con la táctica *algebra*.

Finalmente, la cuarta aproximación, llamada *aproximación por instanciación*, no es de la misma naturaleza que las anteriores. La razón es que la mejora cualitativa obtenida con ella es debida a un recurso interno de Isabelle, llamado *locales*. La instanciación de los locales permite al usuario algunas comodidades notacionales importantes: varias instancias de una misma estructura algebraica (varios anillos, por ejemplo) pueden ser usadas en el mismo fragmento de Isabelle, importando cada propiedad que haya sido probada para ellas. Incluso si la expresividad no es especialmente potenciada en nuestro contexto, las comodidades introducidas por los locales pueden implicar una mejora en la legibilidad y en la construcción de demostraciones que no debe ser subestimada.

En el Capítulo 4, nuestros resultados tienen un doble interés. Por una parte, nos permiten ir más lejos en nuestro problema inicial de probar la corrección de Kenzo. Por otra parte, implican una búsqueda de la relación de nuestro trabajo con el constructivismo. En cambio, hemos ignorado otros problemas relacionados con la extracción de programas. Por ejemplo, no nos hemos ocupado de que los programas extraídos mejoren (en cuanto a su generalidad o eficiencia) a los que están actualmente en uso. La razón principal de esta decisión está basada en las limitaciones de la tecnología actual.

La principal herramienta en el Capítulo 4 es el sistema diseñado por S. Berghofer para extraer programas ML de código Isabelle ([Berghofer, 2003a, Berghofer, 2003b, Berghofer, 2004]). Debido a la riqueza de la lógica de orden superior, no está claro que cualquier fragmento de Isabelle pueda ser utilizado para extraer código de él. Esto da lugar a interesantes problemas en las limitaciones de la aplicación de Berghofer y también en la comparación con otros sistemas basados en matemáticas constructivas (Coq [Bertot and Castéran, 2004] es la alternativa más cercana). En lugar de afrontar este problema con un espíritu *teórico*, tratamos de seguir un camino más *práctico*: estudiar la aplicación del programa de Berghofer a algoritmos del Álgebra Homológica. Se puede observar que la mayor parte de los teoremas interesantes (como se documentó en el Capítulo 2) tienen una propiedad reseñable: en el enunciado, un objeto (estructura algebraica, morfismo, reducción, ...) es definido, y alguna de sus propiedades es hecha explícita. Hemos definido este tipo de enunciado como *enunciado constructivo*. La clave es que entonces el código puede ser extraído del enunciado (o, siendo más precisos, de la definición implícita en el enunciado), sin hacer ninguna referencia a la demostración. Como la demostración no es relevante para el código extraído (excepto en la cuestión esencial de asegurar su corrección, por supuesto), la lógica empleada en la demostración queda desligada del mecanismo de extracción. En particular, la demostración puede ser *no constructiva*.

Para reducir la complejidad del código y de las demostraciones, exploramos este delicado problema con un ejemplo más elemental, por medio de un programa simple en Common Lisp que calcula el primo siguiente a un número natural dado como argumento. Aun siendo simple, este ejemplo no es trivial: es precisamente el mismo ejemplo que Markov eligió ([Markov, 1971]) para ilustrar su generalización de las matemáticas constructivas (conocida hoy en día como *matemáticas constructivas recursivas*). En lugar de intentar demostrar la corrección de ese programa por medio de Isabelle (aparte de las dificultades teóricas relacionadas con el constructivismo, deberíamos afrontar el problema de vincular Common Lisp y ML), hemos reformulado el problema para establecerlo como un enunciado constructivo. Entonces, un programa ML equivalente al inicial ha sido obtenido con ayuda de la aplicación de Berghofer, mientras que el teorema ha sido demostrado usando un argumento de "reductio ad absurdum", lo cual nos da una idea de su no constructivismo. Todos estos resultados, bastante costosos en número de líneas en Isabelle, equivalen a *programar* en Isabelle (además de las habituales tareas de *especificar* y *demostrar* en Isabelle). Las ideas previas han sido aplicadas después a un caso sencillo en Álgebra Homológica: la composición de dos morfismos.

En resumen, la memoria contiene bastantes propuestas meditadas con el objetivo de mecanizar el razonamiento en Álgebra Homológica y obtener programas de Álgebra Computacional certificados para este área. En particular, hemos destacado la importancia del proceso de abstracción en matemáticas mecanizadas, y hemos introducido varios esquemas conceptuales que podrían ser reutilizados en diversas áreas de aplicación. Otro resultado de nuestra investigación es mostrar las limitaciones de ciertas aproximaciones, y establecer los puntos difíciles en el trabajo que queda por hacer. Queda claro que la investigación no está concluida: asegurar que nuestras propuestas son realmente reutilizables es necesario para avanzar en la demostración del Lema Básico de Perturbación, y también habrá que aplicar los métodos de extracción a fragmentos de los algoritmos más complejos. Explicamos nuestros proyectos futuros en la siguiente (y última) sección.

Trabajo Futuro

En primer lugar, sería necesario comparar nuestros hallazgos con otros enfoques. Y esto no tanto con respecto a otras soluciones a nuestro mismo problema (nuestro trabajo, en este sentido, es bastante innovador y original, en nuestra humilde opinión), como respecto a otras herramientas alternativas a las utilizadas hasta la fecha por nosotros. La herramienta más conveniente para comenzar esta comprobación sería el sistema Coq [Bertot and Castéran, 2004]. Para ello, podría interesar el basarse en el sistema FOCAL [Foc, 2005], que proporciona un entorno integrado para la especificación, la programación (por medio del lenguaje de programación OCAML [Leroy et al., 2005, Boulmé et al., 1999]) y la demostración (por medio de Coq).

Pero el trabajo más evidente que queda por realizar es completar la segunda parte de la demostración del Lema Básico de Perturbación. Consideramos que nuestras propuestas serán suficientes para este fin. Quizá la cuestión abierta más importante sea conocer si nuestro tratamiento del *problema de la igualdad* es suficiente. Hasta ahora, la igualdad entre morfismos ha sido tratada (por medio de la noción de *compleción*) de tal forma que se ha hecho innecesario *sobrecargar* la definición de la igualdad en Isabelle. En situaciones donde la igualdad se extiende a estructuras algebraicas, quizás la sobrecarga u otras ideas técnicas o conceptuales podrían ser necesarias.

Posteriormente, la *primera parte* de la demostración debería ser considerada. Trata con una *serie funcional*, y parece claro que cierta clase de proceso inductivo debería ser introducido para manejar esta estructura infinita. Aunque Isabelle soporta esta clase de razonamiento, pueden aparecer cuestiones complicadas, relacionadas tanto con la extracción de código como con los aspectos de fundamentos.

Efectivamente, la extracción de código ML a partir de demostraciones rutinarias en Álgebra Homológica (como las que aparecen en la segunda parte de la demostración del Lema Básico de Perturbación) parece estar resuelta por medio de la aplicación de Berghofer y nuestra técnica de *enunciados constructivos*. Sin embargo, la serie puede presentar problemas de terminación, y es más difícil, sin una mayor investigación, prever si la tecnología actual será suficiente. Explicaremos este punto con un poco más de detalle.

Las demostraciones en el Capítulo 2 referentes a series están basadas en el hecho de que las series funcionales formales definen un morfismo, ya que cuando son aplicadas a cada elemento se convierten en una suma *finita*. Esto depende de la noción de endomorfismo *localmente nilpotente*. En la versión presentada en esta memoria (la usual en la literatura) la situación es muy similar al ejemplo aritmético considerado en el Capítulo 4: se sabe que existe una cota, pero no hay información explícita sobre ella. Por lo tanto, demostrar la terminación es problemático y necesitaría el principio de Markov, yendo más allá del constructivismo estricto (y por tanto, quizá, más allá de las capacidades de la aplicación de Berghofer). No obstante, desde un punto de vista pragmático, la definición puede ser transformada en constructiva (dando lugar a la noción de *nilpotencia local constructiva*), requiriendo una función explícita que calcule el grado de nilpotencia de cada elemento. Esta nueva definición es aplicable a cualquier instancia del Lema Básico de Perturbación que haya sido utilizada en Kenzo (y, hasta donde sabemos, en cualquier aplicación en Álgebra Homológica algorítmica o Topología Algebraica), mostrando una vía para la extracción de código en este marco (no muy diferente de nuestro tratamiento en el Capítulo 4 del ejemplo de los números primos).

Esta pequeña discusión señala claramente que el problema de la naturaleza fundacional de la Topología Algebraica (si puede ser presentada como estrictamente constructivo, recursivamente constructiva o simplemente no constructiva) permanece abierto (esta cuestión ha aparecido en varias ediciones de las conferencias *Mathematics, Algorithms, Proofs*; ver una pequeña reseña sobre este tema en el MAP2005 en [Rubio, 2005]).

Incluso si cualquier problema práctico o teórico es resuelto, y el código ML es extraído del Lema Básico de Perturbación implementado en Isabelle, todavía quedará el salto entre ML y Common Lisp (el lenguaje de programación en el que Kenzo fue escrito). Varias aproximaciones son posibles en este punto. Una de ellas es extraer código de Isabelle a Common Lisp. La otra sería escribir un traductor de ML a Common Lisp (¡certificado!). Ambos proyectos merecen atención, también fuera de nuestro contexto de aplicación.

Y yendo un paso más allá, incluso si se pudiese extraer código Common Lisp, el problema de generar código similar al de *Kenzo* seguirá abierto. En otras palabras, el problema de extraer código eficiente (comparable a Kenzo) no parece posible en el estado actual de la tecnología. Ésta es la razón por la que nuestra aproximación al problema es muy similar a la mostrada en esta memoria con respecto a la demostración mecanizada del Lema Básico de Perturbación. El objetivo no es obtener una demostración automática *completa*, sino incrementar nuestro conocimiento (sobre la misma demostración del Lema Básico de Perturbación, sobre los límites de los sistemas de razonamiento mecanizado actuales, o, en definitiva, sobre la interacción entre las Matemáticas y la Informática). En esta misma dirección, el objetivo real no es demostrar de forma automatizada la corrección de Kenzo (que es un sistema bastante seguro, después de muchos años de exitoso *testing*), sino encontrar métodos que incrementen la fiabilidad de programas informáticos, sin perder eficiencia o aplicabilidad. Los demostradores automáticos de teoremas pueden ser una de las herramientas para este propósito. La principal contribución de esta memoria es haber dado un pequeño paso en esa dirección.

Publicaciones

- **Aransay, J., Ballarin, C. y Rubio, J. . Towards an automated proof of the Basic Perturbation Lemma. En Giménez, P., editor, EACA 2002, Octavo Encuentro de Álgebra Computacional y Aplicaciones, Peñaranda de Duero, Spain, September 2002, páginas 91–95. Universidad de Valladolid.**

  El Lema Básico de Perturbación es una herramienta fundamental para el diseño de algoritmos en Álgebra Homológica. Presentamos el estado de un proyecto para dar una demostración mecanizada del Lema Básico de Perturbación, usando el demostrador táctico de teoremas Isabelle. En particular, presentamos un lema concreto junto a su demostración mecanizada, ilustrando el nivel de abstracción que pretendemos alcanzar.

- **Aransay, J., Ballarin, C. y Rubio, J. . Mechanizing proofs in Homological Algebra. En Zimmer, J. and Benzmüller, C., editors, Calculemus Autumn School 2002: Poster Abstracts, volume SR-02-06, páginas 13–19. Universität des Saarlandes.**

  En los últimos años, ha habido un creciente interés en el Cálculo Simbólico en Álgebra Homológica y en Topología Algebraica. Una herramienta esencial para el diseño de algoritmos en este área es el Lema Básico de Perturbación. Aquí se presenta un proyecto para construir una demostración mecanizada de este lema por medio del demostrador táctico de teoremas Isabelle. Además, un modelo de representación para las estructuras algebraicas que han de ser implementadas en nuestro proyecto es propuesto.

- **Aransay, J., Ballarin, C. y Rubio, J. . Deduction and Computation in Algebraic Topology. En IDEIA 2002, IBERAMIA 2002, I Taller Iberoamericano sobre Deducción Automática e Inteligencia Artificial, Sevilla, Spain, October 2002, páginas 47–54. Universidad de Sevilla.**

  En este artículo se presenta un proyecto para desarrollar una demostración asistida por ordenador del Lema Básico de Perturbación. Este lema es uno de los resultados centrales en Topología Algebraica algorítmica, y obtener una demostración mecanizada de él sería un primer paso para incrementar la fiabilidad de muchos sistemas de Cálculo Simbólico en este área. Se describen algunas técnicas para codificar las estructuras algebraicas necesarias en el demostrador de teoremas Isabelle, y se incluye una secuencia de lemas de alto nivel diseñados para alcanzar dicha demostración mecanizada.

- **Aransay, J., Ballarin, C. y Rubio, J. . Towards a higher reasoning level in formalized Homological Algebra. En Hardin, T. and Rioboo, R., editors, Calculemus 2003, 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Rome, Italy, September 2003, páginas 84–88. Aracne Editrice S.R.L.**

Introducimos una posible solución para algunos problemas encontrados cuando se trata de mecanizar demostraciones de teoremas en Álgebra Homológica: cómo tratar con funciones parciales en una lógica de funciones totales y cómo conseguir un nivel de abstracción que permita al demostrador trabajar con morfismos de forma ecuacional.

- **Aransay, J., Ballarin, C. y Rubio, J. . Four approaches to automated reasoning with differential algebraic structures. En Buchberger, B. and Campbell, J. A., editors, AISC 2004, 7th International Conference on Artificial Intelligence and Symbolic Computation, Linz, Austria, September 2004, volumen 3249 de Lecture Notes in Artificial Intelligence, páginas 222–235. Springer.**

  Al implementar una demostración del Lema Básico de Perturbación (un resultado central en Álgebra Homológica) en el demostrador de teoremas Isabelle uno se encuentra con problemas tales como la implementación de estructuras algebraicas, funciones parciales en una lógica de funciones totales, o el nivel de abstracción en demostraciones formales. Diferentes aproximaciones encaminadas a resolver estos problemas son evaluadas y clasificadas de acuerdo a características tales como el grado de mecanización alcanzado o la correspondencia directa con las demostraciones matemáticas. Basándonos en este estudio, proponemos un entorno para futuros desarrollos en Álgebra Homológica.

- **Aransay, J., Ballarin, C. y Rubio, J.. Extracting computer algebra programs from statements. En Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A., editors, EUROCAST 2005, 10th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 2005, volumen 3643 de Lecture Notes in Computer Science, páginas 159–168. Springer.**

  En este artículo se presenta una aproximación para la obtención de programas correctos a partir de especificaciones. La idea es extraer código de las definiciones que aparecen en los enunciados que han sido demostrados mecanizadamente con la ayuda de un asistente a la demostración. Esta aproximación ha sido hallada al demostrar la corrección de ciertos programas de Álgebra Computacional (para la Topología Algebraica) usando el asistente Isabelle. Para facilitar la comprensión de nuestras técnicas, las ilustramos con algunos ejemplos sobre aritmética elemental.