ADVANCES IN MATHEMATICS 104, 1–29 (1994)

The Computability Problem in Algebraic Topology

FRANCIS SERGERAERT

Institut Fourier, B.P. 74, 38402 St. Martin d'Hères Cedex, France

1. INTRODUCTION

Alonzo Church [CHR] and Alan Turing [TRN] proved in the middle of the thirties that there cannot exist a general algorithm determining whether a mathematical statement is true, false, or undecidable.

This at first sight negative result has eventually become one of the most productive ones in the history of mathematics, at several levels.

At first, and this has been immediately noted, this result ensures the durability of the mathematical job, which is not inconsiderable. No finite "system," in the broadest sense of the term, can *potentially* cover the whole mathematical field. So it can be considered that Church and Turing have proved that mathematicians will always be able to discover new playing fields and their creativity will never die for lack of study areas.

Hilbert's dream actually was a dream. But a new subject for research was thus opened: studying what mathematical theories are in fact open to algorithmic processes. A concrete example will help us to explain this idea. Novikov proved the nonexistence of a general algorithm for solving the word problem in a finitely presented group. But on the contrary Tartakovskii proved that this problem is solvable for a large class of groups; for example, Magnus had proved before Novikov's work the existence of a solution for a group with one relation. More explanations and references can be found under "Free Groups" in [EDM].

This paper is devoted to such a result. Here it is proved that almost every reasonable computability problem in homological algebra and algebraic topology has a positive solution. There remains a unique limitation, as frequently in algebraic topology: simple connectivity hypotheses must often be satisfied; otherwise it is easy to find problems the noncomputability of which results from Novikov's theorem. For example, there does not exist a general algorithm allowing us to decide whether a finite simplicial complex is simply connected.

After having stated this restriction, all the homotopy, homology, K-theory, ..., groups that fill the algebraic topology books "could be" computed on a machine and many others as well.

0001-8708/94 $9.00

In this connection, two earlier results have to be examined for comparison. Edgar Brown [BRW] proved the computability of the homotopy groups of finite simply connected simplicial sets. He obtained this result by programming Postnikov's tower, but he encountered some difficulties which we shall examine in due time (Section 9) because of the hugeness of the $K(\pi, n)$'s. Here these difficulties are overcome by a quite new method, the *functional coding* method, which will be outlined later in this introduction. This method is so powerful that on one hand the class of the problems with a computable solution is at once significantly enlarged and, on the other hand, it is now quite sensible to get down to work and actually program on a computer the algorithms whose existence is thus proved.

Sullivan's so-called *minimal model* method [SLL] allows us to solve many computability problems too; however, the computability problem of the *torsion components* remains unanswered. Various attempts have been made to cover the torsion component case too, but as far as we know, no general result has yet succeeded in finding the hoped for extension.

We obtain here a solution for this problem. Admittedly its nature is strange, so strange that the professionals' agreement seems hard won. We explain in this paper how to assign to a simplicial set a *finite* object, precisely one that can be coded as a *finite bit string*, and that contains its whole homotopy type in an effective way. In fact this cannot be applied to just any simplicial set. On one hand it must be simply connected (or satisfy a nilpotency condition). On the other hand it must be a simplicial set *with effective homology*. But we prove in Section 10 that every "reasonable" simply connected simplicial set has effective homology. For example, $\Omega^2(K(\mathbb{Z}_3, 7) \vee P^4\mathbb{H} \vee \Omega^2 S^6) \vee S^2$ is "reasonable"; it is a simplicial set with effective homology and the object assigned to it is a *finite* bit string allowing us to *compute any one* of its homotopy groups or Postnikov invariants.

What is *functional coding*? This question brings us back to Church. In order to prove the nonexistence of the algorithm hoped for by Hilbert, Church had to create a very clever machine model, where the algorithms are quite capable of working on algorithms in order to make algorithms, and so on. This is the *λ-calculus*. Church's idea was to find a contradiction of type "the liar's paradox" similar to the one used by Gödel for his incompleteness theorem.

Hence Church invented the λ-calculus in order to prove a *negative* result, probably without having any idea of the *positive* applications which would be later obtained! Indeed some computer scientists of the sixties were curious enough to examine whether Church's model could give concrete applications for actual machines. It was the time when McCarthy created the Lisp programming language [MCR]. The λ-calculus, as formal mathematics, needs *terms* whose length grows very quickly. McCarthy's work

therefore consisted in examining the construction details of the λ-calculus and overcoming this problem so as to get a concretely usable programming language. At the beginning this work gave rise only to skepticism but eventually has been recognized as of the greatest relevance: today we cannot keep count of the important if not essential applications of the Lisp language. Thus it is by far the most used programming language in the artificial intelligence field; for applications closer to traditional mathematics, let us point out that very powerful symbolic computing software such as Reduce, Macsyma, and Scratchpad rest on a Lisp base. The results explained in this paper will perhaps expand the Lisp application field still more.

Almost all the objects considered in this paper will be coded as algorithms open to processing by other algorithms. The theoretical justification consequently rests on Church's work; and the assertion stated earlier about the feasibility of these algorithms rests on Lisp programming experience. So that we see that undoubtedly Church's work for proving a negative result will continue to give positive consequences.

Church and Turing will again be useful for examining another aspect of the present work. Their contemporaries had the biggest difficulties in admitting that systems as elementary as the Turing machine or the *λ-calculus* could actually contain everything that is possible to do on any machine. However, experience showed they had the right point of view. For example, the existence of a *universal machine*, proved by Turing and which greatly surprised the mathematicians of his time, results from this strategy. And the whole of modern computer science rests on this idea; this story proves that often *simplicity = efficiency*.

In so speaking we want to explain to the homological algebra professionals that they will find in this paper neither a new exact sequence with magical properties nor a new super spectral sequence to be added to their already rich (and wonderful!) toolbox. On the contrary it is a question of reducing everything that is done in homological algebra to *mechanisms as elementary as possible*. We will show that any task in homological algebra is nothing other than a repeated application of two elementary operations: adding to or removing from a chain complex an acyclic direct summand. See the *reduction* notion in Section 6. For example, a homotopy equivalence is the succession of such an addition and such a subtraction. We strip down in this way all exact and spectral sequences of our folklore. Once this is carried out, the computability results can be obtained very easily.

We must also explain that the techniques described in this paper diminish in no way the importance of the "classical" work done otherwise. They only supplement them in one direction: obtaining computing algorithms from already existing techniques. All the existing exact and spectral sequences remain useful and even essential.

Since these results have been announced [SRG], the author has constantly been asked whether he was able to find in this way new homology or homotopy groups. This question is examined in more detail in Section 10. Here we only recall the long time interval between the genesis of Church and Turing's works and the fantastic applications that are made of them today. A little patience will therefore probably be needed, but the field thus opened seems fascinating.

Several discussions with Claude Quitté, Julio Rubio and Martin Tangora have been very useful and they are warmly thanked. The excellent book [HDG] is highly recommended for the very interesting description which is done about the birth of computer science with Turing of course and also many others; special thanks are due to Yvon Siret (Computer Center at Grenoble University) who called my attention to this marvelous book at the best possible time[1].

2. Machine

In order to justify theoretically the functional coding technique, we need a model for the notion of *machine*, such that there is no difference between the notions of *program* and *data*. This was carried out for the first time during the thirties by some logicians (Church, Kleene, Rosser, ...) who then invented what is now called the *λ-calculus*. The basic reference for the λ-calculus is still [CHS]; a convenient and more recent reference is [HRM, Sect. 31], which gives a fast survey of the essential ideas of the λ-calculus.

In this text, very elementary notions about the λ-calculus are quite sufficient; they are explained now.

A *machine* (or *automaton*) is a triple $(\mathscr{T}, \mathscr{U}, \rho)$ where:

$\mathscr{T}$ is a countable set, the set of *terms* of the λ-calculus;

$\mathscr{U}$ is a subset of $\mathscr{T}$; it is the set of terms in the λ-calculus that are *in normal form*;

ρ is a function: $\mathscr{T}_\rho = \mathscr{T} - \mathscr{U} \to \mathscr{T}$, the *elementary reduction operator* of the *λ-calculus*.

The set $\mathscr{T}$ must be understood as the set of possible *states* of the machine during a computation; and $\mathscr{U}$ is the set of *final* states. The

[1] *Note added in proof.* Since this paper has been accepted (1990) for publication in Advances in Mathematics, the Effective Homology theory has made much progress. On one hand, the concrete programming work has been ended for homology groups of iterated loop spaces and allowed us to reach new unknown homology groups. On the other hand, the "perturbation lemma" has been recognized as the ideal tool to organize the effective homology versions of the various spectral sequences. If interested, please ask the author for preprints (E-mail address: sergerar@imag.fr).

operator ρ defines an elementary step of machine operation; therefore ρ is defined only on the set of non-final states $\mathscr{T}_\rho$. The set $\mathscr{U}$ can also be considered as the set of *machine objects* ($\mathscr{U}$ means *universe*); in fact it is later explained that the elements of $\mathscr{U}$ can be understood on one hand as *programs*, and on the other hand as *data*. Note that $\mathscr{U}$ is a subset of the countable set $\mathscr{T}$, so that $\mathscr{U}$ is countable too.

The function ρ allows us to define another function $\bar{\rho}: \mathscr{T} \to (\mathscr{U} \coprod \{?\})$ which describes the final state, when it is defined, of the machine when it is started from the state $t \in \mathscr{T}$; the function $\bar{\rho}$ is defined as follows: let $t \in \mathscr{T}$ be a term of the λ-calculus; then one and only one event among both of the following can happen:

(a) there exists an integer $n \in N$ such that for any any $m < n$, $\rho^m t \in \mathscr{T}_\rho$; furthermore $\rho^n t \in \mathscr{U}$; then we define $\bar{\rho} t = \rho^n t$; note that n can be null, and in this case $t = \bar{\rho} t \in \mathscr{U}$;

(b) for every $n \in \mathbb{N}$, $\rho^n t$ is defined and is an element of $\mathscr{T}_\rho$; in that case we define $\bar{\rho} t = ?$.

In case (a), the machine stops after n elementary steps; the final result (*output*) of the computation is $\bar{\rho} t$. In case (b) the computation will never stop and the result must be considered as *undefined*.

The abovementioned logicians proved that it is possible to choose $\mathscr{T}$, $\mathscr{U}$, and ρ and define a theoretical machine which is equivalent in a natural sense to any other one [CHS]. The construction of $\mathscr{T}$, $\mathscr{U}$, and ρ is quite simple (and very nice!) but too technical to be explained here; it will not be used in this text. The essential benefit of this method is that in this way there is no difference between *programs* and *data*; we shall even see that any object in $\mathscr{U}$ can be considered as a *program* or as a *datum* on which any program can work (but of course the final result could be undefined). This is an important difference with respect to the other modeling methods (see for instance [AHU, Chap. 1]).

More precisely if we want to consider an object p in $\mathscr{U}$ as a *program* capable of working on a *datum* d (in $\mathscr{U}$ as well), we have to work as follows: in general, if $a, b \in \mathscr{T}$, then (ab) denotes another element in $\mathscr{T}$ which is constructed from the terms a and b in $\mathscr{T}$ by a very simple process. And now if p and $d \in \mathscr{U}$, p considered as a program and d as a datum, then the *result* of the execution of the program p on the datum d is $\bar{\rho}(pd) \in \mathscr{U} \coprod \{?\}$; of course, this result must be considered as actually defined only if $\bar{\rho}(pd) \in \mathscr{U}$. We say that d is the *input* of the program and $\bar{\rho}(pd)$ is the *output*.

More generally the result of the execution of $p \in \mathscr{U}$ on the data $d_1, ..., d_m \in \mathscr{U}$ is $\bar{\rho}(\cdots((pd_1)\, d_2)\cdots d_n)$ which is more simply denoted by $\bar{\rho}(pd_1 \cdots d_n)$.

2.1. DEFINITION. A *type* is a subset of $\mathscr{U}$.

Let *TYPES* denote the set of types so that $TYPES = \mathscr{P}(\mathscr{U})$. Here we appreciably move away from the habits of the logicians and the computer scientists; see for instance [STL, Chap. 4; GRR]; usually a type is in some way a machine object (for a computer scientist) or some *recursive* object (for a logician); on the contrary, here a type is a "mathematical" notion.

We have, and this is essential in this text, a binary operator "$\to$" on *TYPES*, defined as follows; let $A, B \in TYPES$; then

$$\to(A, B) = \{\varphi \in \mathscr{U} \text{ st } \forall t \in A, \bar{\rho}(\varphi t) \in B\}.$$

We frequently denote $\to(A, B)$ more simply by $(A \to B)$ or even $A \to B$ if there is no ambiguity. In other words, $\to(A, B) = (A \to B) = A \to B$ is the set of "programs" that if one gives them as input an element of A, then they return as output an element of B.

The following types are defined and can be used as usual:

Bool $= \{true, false\}$ is a type with two elements that are the results of the *predicates*.

Z is the countable set of machine objects used for coding the elements of the mathematical $\mathbb{Z}$; the usual operators $+, -, *, \ldots$ are objects in $\mathscr{U}$.

$\mathbf{N} \subset \mathbf{Z}$ corresponds to the elements of $\mathbb{N} \subset \mathbb{Z}$.

List is a set of objects in $\mathscr{U}$ which represent the *lists* (finite sequences) of elements in $\mathscr{U}$; some operators allow us to get the length of a list, its elements, to construct a list with given elements, and so on. If $x_1, \ldots, x_n$ are objects in $\mathscr{U}$, the list whose elements are $x_1, \ldots, x_n$ is denoted by $[x_1, \ldots, x_n]$; note that $[x_1, \ldots, x_n]$ is an element of $\mathscr{U}$ as well.

If $A_1, A_2, \ldots, A_n$ are types, the type $A_1 \times A_2 \times \cdots \times A_n$ consists of the lists whose length is n and whose i^{th} element is in A_i. If $x \in A_1 \times A_2 \times \cdots \times A_n$, let $\pi_i x = \pi_i(x) = \bar{\rho}(\pi_i x)$ denote the i^{th} element of x; note that $\pi_i \in (\mathbf{List} \to \mathscr{U} \coprod ?)$ because $\pi_i x$ is not defined if the list x is not long enough. We have the rather strange relations $\mathscr{U} \times \mathscr{U} \times \mathscr{U} \subset \mathscr{U}$, $(\mathscr{U} \to \mathscr{U}) \subset \mathscr{U}$, and so on. In fact *anything* is included in $\mathscr{U}$ and this justifies the choice of the letter $\mathscr{U}$ for *universe*. However, $\mathscr{U}$ is a coutable set.

3. Coding

3.1. Definition. Let E be a ("mathematical") set. A *coding* for E is a pair $(\mathbf{T}_E, \chi_E)$ where $\mathbf{T}_E \in TYPES$ is the coding set and $\chi_E : \mathbf{T}_E \to E$ is the coding function.

If $t \in \mathbf{T}_E$ and $x = \chi(t) \in E$, it must be understood that t is the machine *coding* of the element x of E. No special property is assumed about the

coding function $\chi_{\mathbf{E}}$. In particular $\chi_{\mathbf{E}}$ can be non-injective. This will always happen when coding is of *functional type*, that is, when $\mathbf{T}_{\mathbf{E}}$ is a type $(A \to B)$; then t and t' can be two different elements of $\mathbf{T}_{\mathbf{E}}$ (two different "programs") but such that for any $a \in A$, the equality $\bar{\rho}(ta) = \bar{\rho}(t'a)$ holds; in other words t and t' always return the same output if the same input is given; then, if $\chi_{\mathbf{E}}(t)$ is deduced from the function induced by t from A to B (of course this is the usual way), we get the relation $\chi_{\mathbf{E}}(t) = \chi_{\mathbf{E}}(t')$: two distinct but equivalent programs code the same element of E.

Furthermore, $\chi_{\mathbf{E}}$ can be non-surjective; this precisely allows us to define very interesting new sets; if $\chi_{\mathbf{E}}: \mathbf{T}_{\mathbf{E}} \to E$ is a coding for the elements of E, let E_{eff} denote the subset $im(\chi_{\mathbf{E}})$ of E; it is the set of *effective* or *recursive* elements of E for the coding $\chi_{\mathbf{E}}$.

For example, let us consider $E = \{f: \mathbf{N} \to \mathbf{N}\}$. We can take as the coding set $\mathbf{T}_{\mathbf{E}} = \to(\mathbf{N}, \mathbf{N}) = (\mathbf{N} \to \mathbf{N})$, in other words the set of those programs on our machine such that if their input is an integer, or better the code of an integer, the output is an integer too. Then the natural coding function $\chi_{\mathbf{E}}: \mathbf{T}_{\mathbf{E}} \to E$ is neither injective nor surjective and its image E_{eff} is the set of recursive functions of the logicians. Here, it is obvious that $\chi_{\mathbf{E}}$ cannot be surjective: whatever coding is chosen, the coding set $\mathbf{T}_{\mathbf{E}}$ is a subset of $\mathscr{U}$ which is countable whereas the set E is not; this is a frequent situation.

Now the essential point which gives much interest to the "effective homology" theory is that any element of $\mathscr{U}$, even if it looks like a program more than a datum, can be used by another program as input for producing as output a new "program" which codes a new mathematical object.

The very simple following example is typical of what it is very easy to do in λ-calculus: there exists a λ-calculus object (in $\mathscr{U}$) which can be called *comp* capable of working on two elements f and g of $(\mathbf{N} \to \mathbf{N})$ and producing the composite h of f and g, so that *comp* belongs to $(((\mathbf{N} \to \mathbf{N}) \times (\mathbf{N} \to \mathbf{N})) \to (\mathbf{N} \to \mathbf{N}))$ and codes the corresponding element of $(\mathbf{N}^{\mathbf{N}})^{\mathbf{N}^{\mathbf{N}} \times \mathbf{N}^{\mathbf{N}}}$.

Not many practical programming languages allow such work on a machine concretely and easily. The most efficient one in this field is the LISP language; there is a very simple explanation of this assertion: LISP is directly inspired by the λ-calculus, which in turn was invented for solving such problems at a theoretical level. We suggest you read in [STL] how carefully the problems of identifier scope have been studied in order to get very elegant solutions for such programming problems (see Chapter 3); in fact the example mentioned above of the object *comp* is used by Steele to illustrate how simple and efficient *Common-LISP* is for constructing *algorithms that can construct algorithms* and even *algorithms that can construct algorithms that can construct algorithms*, and so on.

4. A Didactical Example of Functional Coding

The usual coding of a simplicial complex consists in entering into the machine the list of its vertices, of its edges, of its 2-simplices, and so on with a structure allowing us to find the necessary information again for any use. But of course it is impossible in this way to code simplicial complexes with an infinite number of simplices.

The problem of coding in a machine such simplicial complexes even seems so strange that it has probably not been studied yet. However, the standard methods in algebraic topology commonly use such complexes, for example, loop spaces, classifying spaces, homotopy fibers, ... Edgar Brown met this problem in his famous study about the computability of homotopy groups [BRW] and overcame it in the following way: he proved that the infinite simplicial complexes which he had to work with could be replaced by finite complexes with the same homotopy type up to some given dimension. This method is rather heavy at a theoretical level—it has never yet been applied in other situations—and is definitely out of reach for practical computations: Brown's "finite" complexes are so bulky that it is of course impossible to put them into any actual machine.

We present in this section an example which must be considered as didactical: the machine coding of the loop space of a simplicial complex. It will not be used later but we think it illustrates quite well the coding potential that is given by the functional technique. It can be proved that it is possible to code in the same way the *recursive simplicial sets* and so to get a simple and powerful method for proving the computability of many very interesting homology and homotopy groups.

We slightly modify the notion of a simplicial complex in order to adapt it to our machine situation.

4.1. Definition. A *simplicial complex* is a pair $K=(V, S)$ where $V \subset \mathscr{U}$ and S is a set of finite subsets of V satisfying the following conditions:

(1) if $v \in V$, then $\{v\} \in S$;

(2) if $\sigma \in S$ and $\sigma' \subset \sigma$, then $\sigma' \in S$.

V (resp. S) is the set of *vertices* (resp. *simplices*) of K.

We take the vertices of a simplicial complex among the objects of our machine. Note that we de not ask that the set of vertices V be finite; for example, the *simplex freely generated by* $\mathscr{U}$ is the simplicial complex whose vertex set is $\mathscr{U}$ (it is countable) and the simplex set is $\mathscr{P}_{fin}(\mathscr{U})$ (countable too); let us denote it by $\Delta^{\mathscr{U}}$. Therefore, in our setting, any other simplicial complex is a subcomplex of $\Delta^{\mathscr{U}}$.

Let SC be the set of all simplicial complexes. We now define a *functional* coding for the elements of SC which will allow us, if K is a recursive simplicial complex, to code its loop space as well. Better, we see the existence of an object $omega \in \mathscr{U}$ capable of working on the code k of a recursive simplicial complex K and producing a code $\bar{\rho}(omega\ k)$ for the loop space of K. The author has an explicit and operational expression of *omega* in Common-Lisp at any interested reader's disposal; the construction of such an operator is a simple exercise for a Lisp programmer.

We now define the functional coding for the elements of SC. Let **SC** be the set of those elements $\tau \in (\mathbf{List} \to \mathbf{Bool})$ satisfying the following property: if l_1 and $l_2 \in \mathbf{List}$, if $l_1 \subset l_2$ (this means that any element of l_1 is an element of l_2), and if $\bar{\rho}(\tau l_2) = true$, then $\bar{\rho}(t l_1) = true$. There we have the essential property of the vertex sets of a simplex in a simplicial complex.

Now we can naturally define the coding map χ: $(\mathbf{SC} \to SC)$. Let τ be an element of **SC**. Then a simplicial complex $K_\tau = (V_\tau, S_\tau)$ in SC can be defined as

$$V_\tau = \{v \in \mathscr{U} \text{ st } \bar{\rho}(\tau[v]) = true\};$$

$$S_\tau = \{\{v_1, ..., v_n\} \subset \mathscr{U} \text{ st } \bar{\rho}(t[v_1, ..., v_n]) = true\}.$$

We recall (see Section 2) that $[v_1, ..., v_n]$ is the list object in $\mathscr{U}$ that codes the list of objects of $\mathscr{U}$ consisting of the objects $v_1, ..., v_n$ of $\mathscr{U}$.

Therefore we define $\chi(\tau) = K_\tau$; the function χ is neither injective nor surjective, and we write $SC_{eff} = im(\chi)$; it is the set of *recursive* simplicial complexes (with respect to this coding). For example, the element *id-true* in $(\mathbf{List} \to \mathbf{Bool})$, which always replies *true*, codes $\Delta^{\mathscr{U}}$: $\chi(id\text{-}true) = \Delta^{\mathscr{U}}$; just as the element *id-false* codes the empty complex; the element $\tau_n \in (\mathbf{List} \to \mathbf{Bool})$ which replies *true* if and only if the argument list has less than $n+2$ different elements codes the n-skeleton of $\Delta^{\mathscr{U}}$. The finite simplicial complexes of course all satisfy the recursiveness property.

The set **SC** is countable but SC is not, so that there are many non-recursive simplicial complexes.

Let $K = (V, S) \in SC$ and $v_0 \in V$. We are now going to define a new simplicial complex $\Omega(K, v_0)$, a simplicial version of the loop space of K, that is, $\Omega(K, v_0)$ has the homotopy type of the usual loop space of the geometric realization of K based in v_0.

So we have to define $\Omega(K, v_0) = (V', S')$.

At first $V' \subset \mathbf{List}$ is the set of those lists $[v_1, ..., v_n]$ satisfying $\{v_0, v_1\}$, $\{v_1, v_2\}$, ..., $\{v_{n-1}, v_n\}$, $\{v_n, v_0\} \in S$. Let us now assume that $l_1 = [v_1^1, ..., v_{n_1}^1]$, ..., $l_m = [v_1^m, ..., v_{n_m}^m]$ are elements of V'; we have to decide whether $\{l_1, ..., l_m\} \in S'$; let us work as follows: let l_i' be the sequence $l_i' = (v_n^i)_{n \in \mathbb{N}} = (v_0, v_1^i, ..., v_{n_i}^i, v_0, v_0, v_0, ...)$; then $\{l_1, ..., l_m\} \in S'$ if and only if for any integer $i > 0$, the relation $\{v_{i-1}^1, v_{i-1}^2, ..., v_{i-1}^m, v_i^1, v_i^2, ..., v_i^m\} \in S$

holds. The lists $l_1, ..., l_m$ are finite so that there are only a finite number of tests to be done. It is easy to see that we have well defined a simplicial complex $\Omega(K, v_0)$.

The intuitive understanding of this definition is not hard: a *vertex* loop in $\Omega(K, v_0)$ is a mapping $[0, \infty[\to K$ which runs through the edges of K but stays at v_0 after some time. A finite set of such vertices spans a simplex of $\Omega(K, v_0)$ if and only if the vertex loops can be interpolated by a barycentric method.

It should now be clear that if we have an object $\tau \in \mathbf{SC}$ coding K, we can find another one τ' coding $\Omega(K, v_0)$; in programming terms, this means that if we have a *program* τ capable of testing whether some list is an element of S, then it is possible to construct another program τ' which will tell you whether some list is an element of S'. In fact the transformation $\tau \mapsto \tau'$ can be *programmed*; in other words there exists an object *omega* $\in$ $((\mathbf{SC} \tilde{\times} \mathscr{U}) \to \mathbf{SC})$ (where $\mathbf{SC} \tilde{\times} \mathscr{U} = \{(\tau, v) \in \mathbf{SC} \tilde{\times} \mathscr{U} \text{ st } \bar{\rho}(\tau[v]) = \mathit{true}\}$) which does the following work: if $(\tau, v_0) \in \mathbf{SC} \tilde{\times} \mathscr{U}$, then $\bar{\rho}((\mathit{omega}\ \tau)\, v_0)$ is a coding for $\Omega(\chi(\tau), v_0)$; so that the following diagram is commutative:

$$\begin{array}{ccc} \mathbf{SC} \times \tilde{\mathscr{U}} & \xrightarrow{\mathit{omega}'} & \mathbf{SC} \\ {\scriptstyle \chi \times id}\downarrow & & \downarrow{\scriptstyle \chi} \\ SC \tilde{\times} \mathscr{U} & \xrightarrow{\Omega} & SC. \end{array}$$

The low part of the diagram exists only at a mathematical level; on the contrary, the high part can be entirely implemented into a machine; *omega'* is the function defined by $\mathit{omega}'(\tau, v_0) = \bar{\rho}((\mathit{omega}\ \tau)\, v_0)$; the vertical arrows are coding maps.

This kind of result is conveniently expressed as follows:

PROPOSITION. *An algorithm can be constructed*:

input: (τ, v_0); τ *is a simplicial complex and* v_0 *one of its vertices.*

output: $\Omega(\tau, v_0)$, *the loop space of* τ *based in* v_0.

This kind of statement asks the reader for the interpretation given earlier. From now on we use this kind of statement without giving its translation, which is a simple exercise without any particular interest. The situation is quite similar to Bourbaki's at the end of the book "Set theory": at this time it is understood that every correct mathematical statement and every correct mathematical proof must be translatable into "formal mathematics" but of course this translation is not given anymore! However, there exists a real difference: we claim that it is actually possible to carry out the

algorithms whose existence is stated, in order to compute mathematical objects unknown until now; so that after the mathematician here the programmer has to work; we hope that the example of the machine coding of the functor "loop space," very easy to carry out, will persuade the reader that this aim is quite realistic.

5. Effective Modules and Chain Complexes[2]

5.1. Definition. An *effective* set is an element of $\mathbf{Ens} = (\mathscr{U} \to \mathbf{Bool})$; if $e \in \mathbf{Ens}$, we define $\chi_{\mathbf{Ens}}(e) = \{x \in \mathscr{U} \text{ st } \bar{\rho}(ex) = \mathit{true}\}$.

This definition should be interpreted as follows: first the classical notion of *set* is redefined; from now on a set must be a subset of $\mathscr{U}$. Let *ENS* denote the set of such sets; a coding is then defined for *ENS*; the coding set is **Ens** and the coding map is $\chi_{\mathbf{Ens}}$; we read in this definition that a set e is an algorithm capable of answering the question, "Is x an element of e?" by *true* or *false*; in other words a set is coded by its characteristic function. This kind of translation, always easy to carry out, will not be given from now on. The functional coding technique will be often used in what follows; in such a situation, it is convenient to denote simply by $e(x)$ the element of $\mathscr{U}$ that should be theoretically denoted by $\bar{\rho}(ex)$.

5.2. Definition. An *effective module* is an element of $\mathbf{Mod} = (\mathscr{U} \to \mathbf{Bool})$. If $m \in \mathbf{Mod}$, we define $\chi_{\mathbf{Mod}}(m)$ as the free $\mathbb{Z}$-module generated by $\chi_{\mathbf{Ens}}(m)$.

Some important differences with mathematical habits must be noted. Here a module will always be a *free* $\mathbb{Z}$-module; this point is fundamental; if *effective homology* is to be compared with classical homological algebra, we may say that effective homology consists in reconstructing classical homological algebra after having forbidden any *torsion module*. Indeed, it is well known that most of the difficulties encountered in homological algebra are due to the torsion components in the various groups to be used or computed. These difficulties arise in particular in *computability*: the usual "computing methods" (exact sequences and spectral sequences) of course already give many very interesting results, but very often leave their users stopped in front of *extension problems*, for which new methods must always be invented. We shall see that overcoming this prohibition from using torsion modules is not actually hard and on the contrary we thus obtain very interesting applications. Here let us say only that mechanisms known as rather complicated, such as spectral sequences, get in effective homology a much simpler look and that furthermore all the computability problems are solved at once, without fatigue.

[2] *Note added in proof.* This terminology has been modified in more recent papers: "effective" is now "locally effective" and "computable" is now "effective".

Another difference with the usual modules is that here modules are equipped with an explicit base whose elements are machine objects ($\in \mathscr{U}$); this is important for computability problems. Finally, we only consider modules with respect to the ring $\mathbb{Z}$, but this is quite inessential: all that is explained in this text can be easily extended to modules over *effective rings*; we do not study this question here, but the particular case of $\mathbb{Z}_{(p)}$, the ring of the integers localized at the prime p, has many applications.

5.3. DEFINITION. Let M be an effective module. Let us define the type **El-Mod**(M) as follows; first **Mon**(M) = $\mathbf{Z} \times M$, the set of *monomials* in M, is the set of pairs having the first element in **Z**, the second in the canonical base of M; the set of monomials in M is therefore the union of the *axes* of M; then **El-Mod**(M) = **List**(**Mon**(M)) naturally is the set of expressions of the elements of the module M as $\mathbb{Z}$-linear combinations in the canonical base, in the form of monomial lists. Note that such an expression is not unique and the coding map is never injective; on the contrary it is always surjective.

5.4. DEFINITION. Let $M, N \in$ **Mod**. We define

$$\mathbf{Morph}(M, N) = (M \to \mathbf{El\text{-}Mod}(N)).$$

In other words a morphism with M as source and N as target is coded as an algorithm which, if a generator of M is given as input, returns as output an element of N in the form explained above.

There exists a universal operator which is able to compute the composite of two morphisms. Indeed, any module is a sub-module of the maximal module $\mathscr{U}$ (coded as the function that always returns *true*), and it is easy to construct an object (a program) *comp-morph* in

$$\begin{aligned}&(((\mathscr{U} \to \mathbf{El\text{-}Mod}(\mathscr{U})) \times (\mathscr{U} \to \mathbf{El\text{-}Mod}(\mathscr{U}))) \to (\mathscr{U} \to \mathbf{El\text{-}Mod}(\mathscr{U})))\\ &\quad = ((\mathbf{Mor}(\mathscr{U}, \mathscr{U}) \times \mathbf{Mor}(\mathscr{U}, \mathscr{U})) \to \mathbf{Mor}(\mathscr{U}, \mathscr{U}))\end{aligned}$$

capable of computing the composite of two morphisms given as input. By restriction, if A, B, C are three elements in **Mod**, *comp-morph* can be also considered as an element of $((\mathbf{Mor}(A, B) \times \mathbf{Mor}(B, C)) \to \mathbf{Mor}(A, C))$ able to carry out the composition operation.

Note that the natural coding map defined on **Mor**(A, B) is never injective and is surjective if and only if the module A is finite dimensional.

5.5. DEFINITION. An *effective chain complex* C is an element $C \in (\mathbf{N} \to (\mathscr{U} \times \mathscr{U}))$ satisfying the following conditons:

(a) if $n \in \mathbf{N}$, then $\pi_1(C(n)) = \pi_1(\tilde{\rho}(Cn)) \in$ **Mod**;

(b) if $n \in \mathbf{N}$, then $\pi_2 C(n) \in \mathbf{Morph}(\pi_1(C(n)), \pi_1(C(n-1)))$;

(c) if $n \in \mathbf{N}$ and $n > 1$, then the composite of the morphisms coded by $\pi_2 C(n)$ and $\pi_2 C(n-1)$ is the null morphism. This amounts to saying that $\bar{\rho}$(comp-morph, $[\pi_2 C(n), \pi_2 C(n-1)]$) = comp-morph($[\pi_2 C(n), \pi_2 C(n-1)]$) codes the null morphism from $\pi_1 C(n)$ to $\pi_1 C(n-2)$.

Let **Ch-Comp** denote the type whose elements are the effective chain complexes.

We find in this definition the usual properties of chain complexes. Note that it is possible to code in this way, theoretically and practically, chain complexes that are quite enormous; the following example is interesting: if Π is an effective abelian group (we leave to the reader the definition), then an object $C_* K(\Pi, n) \in \mathscr{U}$ can be easily constructed, that codes the chain complex of the standard simplicial model of the Eilenberg–MacLane space $K(\pi, n)$. Better, there exists an object $C_* K$ in $\mathscr{U}$ which gives $C_* K(\Pi, n)$ as output if an effective abelian group Π and an integer n are given as input.

5.6. DEFINITION. A *finite module* is a non-negative integer. Let **Fin-Mod** denote the type whose elements are the finite modules so that **Fin-Mod** = **N**; the coding map is defined as follows: if n is a non-negative integer, $\chi_{\mathbf{Fin\text{-}Mod}}(n)$ is the free $\mathbb{Z}$-module generated by the integers $1, \ldots, n$ and therefore is canonically isomorphic to $\mathbb{Z}^n$. Note that **Fin-Mod** is not a subtype of **Mod**. However, there exists a conversion operator able to convert an element of **Fin-Mod** into an element of **Mod**.

There does not exist any algorithm capable of deciding whether a module in **Mod** is finite dimensional; this amounts to saying there does not exist any algorithm capable of deciding whether a set defined by its characteristic function is finite or not (or even empty!); the negative answer to this question is well known to the logicians. Therefore it is impossible to construct an algorithm $\dim \in (\mathbf{Mod} \to (\mathbf{N} \coprod \{\infty\}))$ with the hoped-for property. On the contrary there exists of course $\dim \in (\mathbf{Fin\text{-}Mod} \to \mathbf{N})$ with this property; it is simply the identity algorithm! We see that the two coding types are very different.

5.7. DEFINITION. A *computable chain complex* is an element C in $(\mathbf{N} \to (\mathscr{U} \times \mathscr{U}))$ satisfying conditions similar to those earlier stated, with a difference: now, for every $n \in \mathbf{N}$, $\pi_1 C(n)$ must be an element in **Fin-Mod** and the other parts of the definition are modified accordingly.

This definition is justified by the following observation. If n is a natural number, it is easy to construct—and the author has actually constructed—an element H_n in $\mathscr{U}$ which, if a computable chain complex C is given as input, returns the n^{th} homology group of C as output. Indeed, the data allow us to find the dimensions of the useful chain groups and the rest is

classical algorithmic theory. A similar assertion is of course quite false for effective chain complexes, because for such a complex it is impossible to decide whether some chain group is null or not!

From now on, it seems useless to continue to give statements, definitions, and proofs with so many algorithmic details. The translation into this kind of language is a simple exercise without any particular difficulty provided that the essential difference between both possible codings of a set, a module base, and so on, is not forgotten: the first one consists in giving an element list (it is an *extension definition*), the second one consists in giving a characteristic property of an element (*understanding definition*).

6. Homotopy Equivalences and Cones

We recall that all chain complexes here considered are free $\mathbb{Z}$-module complexes.

In classical homological algebra, the notion of *homotopy equivalence*, sometimes called *chain equivalence*, between chain complexes is very often used. This section is devoted to this notion, in order to redefine it in a fairly different although equivalent way, so that solutions for computability problems become much easier.

Let C_* and C'_* be two effective chain complexes; usually a homotopy equivalence between C_* and C'_* is defined as a pair (f, g) where f is a morphism $f: C_* \to C'_*$, g a morphism $g: C'_* \to C_*$, such that there exist a homotopy operator h on C_* and a homotopy operator h' on C'_*, satisfying $id - g \circ f = h \circ d + d \circ h$ and $id - f \circ g = h' \circ d + d \circ h'$.

The topologists have known for a long time that in such a situation it is much better to *include* the homotopy operators in the *homotopy equivalence* data. But this is not enough to overcome frequent difficulties. We now give a simple and typical example of such a difficulty.

Let us consider the following situation. Let A_* and B_* be two chain complexes, and let $\varphi: A_* \to B_*$ be a complex morphism. Then one defines the *cone* chain complex of φ, denoted by $C(\varphi)$, as follows: its n^{th} chain group $C(\varphi)_n$ is the direct sum $A_{n-1} \oplus B_n$ and the boundary operator $d: C(\varphi)_n \to C(\varphi)_{n-1}$ is defined by the matrix $\begin{bmatrix} d_A & 0 \\ \pm\varphi & d_B \end{bmatrix}$. Now let us suppose we have a homotopy equivalence (f, g) between A_* and another complex A'_*. Then we should like to deduce from these data a homotopy equivalence between $C(\varphi)$ and the cone $C(\varphi')$ of $\varphi' = \varphi \circ g : A'_* \to B_*$. But there is already a problem in defining a morphism from $C(\varphi)$ to $C(\varphi)'$; the definition of such a morphism needs a homotopy operator h between id_A and $g \circ f$, and this is already a good reason to include such an operator in the homotopy equivalence data; then the desired morphism from the cone of φ to that of φ' if $F = \begin{bmatrix} f & 0 \\ \pm\varphi h & id_B \end{bmatrix}$. But the difficulties are not ended; we

have another morphism in the other direction which of course if a homotopic inverse of the previous one: $G = \begin{bmatrix} g & 0 \\ 0 & id_B \end{bmatrix}$; the problem is now to exhibit homotopy operators that prove (F, G) actually is a homotopy equivalence; no simple computation can give the solution. Usually one manages in the following way: one looks at both exact sequences of the cones φ and φ', applies the *five lemma* in order to prove that F induces an isomorphism between the homology groups, and a general result [DLD, II.4.3] then implies that F is actually a homotopy equivalence.

This mechanism is clumsy and not at all convenient for any actual computations. We give now another homotopy equivalence definition, very natural, but that solves this kind of problem.

6.1. DEFINITION. A reduction of the complex A_* to the complex A'_* is a triple $\rho = (f, g, h)$ where:

(a) f is a morphism $A_* \to A'_*$;

(b) g is a morphism $A'_* \to A_*$;

(c) h is a homotopy operator (i.e., with degree 1) on A_*;

(d) f, g, and h satisfy the relations:

(d1) $f \circ g = id_{A'}$;

(d2) $f \circ h = 0$;

(d3) $h \circ g = 0$;

(d4) $id_A - g \circ f = h \circ d_A + d_A \circ h$;

(d5) $h \circ h = 0$.

The intuitive interpretation of this relation set will help to understand it. These relations express that A_* is the direct sum of A'_* and a contractible (acyclic) complex with a given contraction. This decomposition is simply $A_* = \ker f \oplus \operatorname{Im} g$.

Indeed (d1) expresses that $\operatorname{Im} g$ is a direct summand in A_*, (d2) and (d3) say that h is a homotopy operator defined on the canonical complement $\ker f$, and (d4) and (d5) say that this operator is a contraction for $\ker f$.

This relation set can seem awkward; however, in all the situations where reductions from complexes to smaller complexes naturally take place, it is always possible to get such relations. A typical example is the Brown theorem also known as the "twisted Eilenberg–Zilber theorem" [BRX]: Weishu Shih gave complete and explicit formulas for the reduction operator from the chain complex of a simplicial geometric fibration to the complex of the corresponding algebraic fibration. It is probably one of the most complicated reduction operators that have ever been constructed. But

this operator just satisfies all the relations (d1) to (d5) (see [SHH], p. 115, Theorem 1]).

6.2. DEFINITION. A *homotopy equivalence* between the complexes A_* and A'_* is a triple $(\hat{A}_*, \rho, \rho')$ where:

(a) A_* is a chain complex;

(b) ρ is a reduction from $\hat{A}_*$ to A_*;

(c) ρ' is a reduction from $\hat{A}_*$ to A'_*.

It is clear that these hypotheses allow us to find very easily a homotopy equivalence between A_* and A'_* again in the usual sense. The converse is true as well; the proof is a rather tedious exercise similar to those in [DLD, II.4]. But we do not need the equivalence between both definitions, so that we prefer to leave it as an excercise for the interested reader.

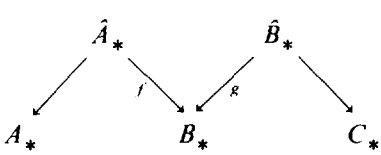
Of course this definition is to be compared with the Grothendieck operator that associates to a projective module its stable isomorphism class; this operator leads to the definition of the object $\tilde{K}_0$ of the base ring; it is enough to translate *projective module* $\rightarrow$ *chain complex*, *free module* $\rightarrow$ *acyclic complex*. So it would be interesting to know whether the other K_i of the algebraic K-theory can have applications in this situation.

6.3. LEMMA. *Let A_* and B_* be complexes, h a contraction of B_* (in other words, h defines a reduction of B_* to the null complex), and φ a morphism $A_* \rightarrow B_*$ (or $B_* \rightarrow A_*$). Then there is a canonical reduction of $C(\varphi)$ (the cone of φ) to A_*.*

This lemma would be trivial if φ was the null morphism; the knowledge of h allows us to solve the general case easily.

6.4. LEMMA. *Let (f, g, h) be a reduction of A_* on A'_*. Then the cones of f and g are canonically contractible.*

6.5. PROPOSITION. *Let $(\hat{A}_*, \rho, \rho')$ be a homotopy equivalence between A_* and B_*, and $(\hat{B}_*, \sigma, \sigma')$ a homotopy equivalence between B_* and C_*. Then these data induce a canonical homotopy equivalence between A_* and C_*.*



Proof. Among other things we have a reduction $f: \hat{A}_* \rightarrow B_*$ and a reduction $g: \hat{B}_* \rightarrow B_*$. By adding to g the null morphism whose source is

$\hat{B}_*$ and target is $\hat{A}_*$, we get a morphism $g'\colon \hat{B}_* \to C(f)$. Let D_* be the cone of g'. Applying both previous lemmas gives a reduction of D_* to $\hat{B}_*$, and then considering the reduction of $\hat{B}_*$ to C_* and the obvious transitivity of the reduction relation, we obtain a canonical reduction of D_* to C_*. But the construction of D_* is quite symmetric so that D_* can be also considered as the cone of $f'\colon \hat{A}_* \to C(g)$ constructed in a similar way. Therefore we also obtain a canonical reduction of D_* to A_*. ∎

Now we examine why the difficulties about the mapping cones explained earlier are cleared up.

6.6. PROPOSITION. *Let $\varphi\colon A_* \to B_*$ be a chain complex morphism, $(\hat{A}_*, \rho, \rho')$ a homotopy equivalence between A_* and A'_*, and (B_*, σ, σ') a homotopy equivalence between B_* and B'_*. Then there exists a canonical homotopy equivalence between the cone of φ and that of the morphism $\varphi''\colon A'_* \to B'_*$ which is induced by these data.*

Proof. First we construct from φ a morphism $\hat{\varphi}\colon \hat{A}_* \to B_*$ by adding to A_* the complementary acyclic complex in $\hat{A}_*$, and adding to φ a null morphism. It is easy to define a reduction of $C(\hat{\varphi})$ to $C(\varphi)$. Furthermore we have a reduction (f, g, h) of $\hat{A}_*$ to A'_* and by using it, it is possible to construct a reduction of the cone $C(\hat{\varphi})$ of $\hat{\varphi}$ to that of $\varphi' = \hat{\varphi} \circ g$; this reduction (F, G, H) is defined by the formulas

$$F = \begin{bmatrix} \pm f & 0 \\ \pm \varphi h & id \end{bmatrix}, \qquad G = \begin{bmatrix} \pm g & 0 \\ 0 & id \end{bmatrix}, \qquad H = \begin{bmatrix} h & 0 \\ 0 & 0 \end{bmatrix}.$$

Then the same procedure gives a homotopy equivalence between the cone of φ' and that of $\varphi''\colon A'_* \to B'_*$. But the homotopy equivalence relation is (effectively) transitive and the proof is complete. ∎

7. HOMOLOGY GROUPS: A NEW DEFINITION

7.1. DEFINITION. Let A_* be an *effective* chain complex. The *effective homology* of A_* is a pair (more precisely a two element list in $\mathscr{U}$) $[HA_*, h]$ where:

(a) HA_* is a computable chain complex;

(b) h is a homotopy equivalence between A_* and HA_*.

As usual in this kind of situation, we also say briefly that HA_* is the effective homology of A_*; the homotopy equivalence between A_* and HA_* is then underlying.

7.2. DEFINITION. A *chain complex with effective homology* is a triple $[A_*, HA_*, h]$ where A_* is an effective chain complex and $[HA_*, h]$ is its effective homology.

No algorithm can compute the effective homology of every effective chain complex, which is furthermore not generally defined (think of finiteness problems). The goal of the previous definitions is not at all of this kind; the question is to define a framework where, if one knows the homology of two objects and if one constructs with these objects a third one, then an algorithm is capable of deducing the homology of the new object. Now we state such a result; it is a simple corollary of the propositions in the previous section.

7.3. THEOREM. *An algorithm can be constructed*:

input: $(A_*, HA_*, B_*, HB_*, \varphi)$; A_* *and* B_* *are effective chain complexes,* HA_* *and* HB_* *are their respective effective homology, and* φ *is a morphism* $A_* \to B_*$.

output: $HC(\varphi)$, *the effective homology of the cone* $C(\varphi)$.

We see that in this way we have a strictly defined framework where the computability problem of the homology of a mapping cone is completely solved. Of course the direct scope of this result is limited; it has been given at the outset in order to explain in a simple situation the precise nature of some techniques which indeed we want to apply in much more interesting situations, when the "classical" situation uses the spectral sequence method. In fact we shall see that the effective homology version of spectral sequence techniques can be obtained by simple recursive use of the previous theorem about cones.

We claim that the object type "chain complex with effective homology" thus constructed affords many advantages:

(a) Thanks to functional coding, the *effective chain complex component* allows us to overcome all the difficulties related to the natural appearance of highly infinite complexes (loop spaces, classifying spaces, ...).

(b) The *computable chain complex component* allows us, if someone actually wants it, and of course it is the motivation of all this work, to obtain the *ordinary* homology groups of the complexes under consideration.

(c) The *homotopy equivalence component* establishes a kind of telephone line between both complexes allowing, if one works inside one of them, to ask the other for some information not available in the first one because of its nature.

(d) This organization is *stable*; we want thus to explain that every "classical" construction process for a new chain complex can be fairly easily reorganized so that, if the data have the format described above, then it produces an *object with the same format*. We think that Theorem 7.3 which describes the computation of the effective homology of a mapping cone, if the effective homology of the components is available, explains this phenomena in a simple case.

(e) In spite of the richness of information in this object type, such an object is still an element of $\mathscr{U}$ and therefore can be coded on an abstract or actual machine as a *finite bit string*.

Functional coding, because of its very nature, uses very little memory. Thus by this organization, the classical problems of the algebraic topologists will enter a machine as more or less big sets of functions which will naturally exploit the classical programming techniques, in particular subprograms and recursiveness; what is quite original is that this program set is itself created by other programs created by other programs, and so on. As far as we know, it is the first time an "ordinary" mathematical theory uses in an essential way and in a very concrete situation the λ-*calculus* techniques which were invented by Church for a highly theoretical reason: proving the nonexactness of the Hilbert conjecture about the existence of a general algorithm deciding whether a statement is true, false, or undecidable.

8. Complex Towers and Spectral Sequences

In this section we want to explain what the spectral sequence notion becomes in effective homology. Here we collect the benefits of the little painful work carried out before:

(a) The *spectral sequence mechanism* will become completely *algorithmic*; in other words, any "classical" spectral sequence (Leray, Serre, Eilenberg–Moore, ...) becomes an algorithm which, if some complexes with effective homology are given as input, returns as output the hoped-for chain complex with effective homology.

(b) The so-called spectral sequence theory will practically disappear! Indeed we explain that the spectral sequence algorithm in this new framework is nothing further than a recursive application of the "cone" algorithm explained in Section 6.

8.1. DEFINITION. A *complex tower* is a sequence (translate: an algorithm $\mathbf{N} \to \ldots)(C_*^p, f^p, T_*^p)_{p \in \mathbb{N}}$ where:

(a) C_*^p is an effective chain complex;

(b) f^p is a morphism $f^p: C_*^p \to T_*^{p-1}$ of degree $p-1$, that is, $f_n^p: C_n^p \to T_{n+p-1}^{p-1}$ and those f_n^p commute with boundary operators;

(c) $T_*^0 = C_*^0$ and T_*^p is the cone of f^p graded so that the inclusion $T_*^{p-1} \subset T_*^p$ is of degree 0.

The index p must be understood as a filtration degree. Now we give two characteristic examples.

8.2. DEFINITION. An *effective bicomplex* is a sequence $(C_*^p, f^p)_{p \in \mathbb{N}}$ where:

(a) C_*^p is an effective chain comples;

(b) f^p is a morphism $f^p: C_*^p \to C_*^{p-1}$ satisfying $f^{p-1} \circ f^p = 0$.

It is easy to associate a complex tower to a bicomplex: take $T_*^0 = C_*^0$, consider the canonical inclusion from C_*^{p-1} into T_*^{p-1}, and deduce from f^p a morphism from C_*^p to T_*^{p-1} which can be called f^p as well.

8.3. DEFINITION. An *effective multicomplex* is a system $(C_q^p, f_q^{p,r})_{p,q,r \in \mathbb{N}}$ where:

(a) C_q^p is an effective module;

(b) $f_q^{p,r}: C_q^p \to C_{q+r-1}^{p-r}$ is a module morphism;

(c) $\sum_{0 \leqslant s \leqslant r} f_{q+s-1}^{p-s,r-s} \circ f_q^{p,s} = 0$ for every $p, q, r \in \mathbb{N}$.

A bicomplex is a particular case of multicomplex. Furthermore, a complex tower ban be canonically associated to any multicomplex and vice versa so that both notions in fact are equivalent. If $(C_q^p, f_q^{p,r})_{p,q,r \in \mathbb{N}}$ is a multicomplex and p fixed, $(C_*^p, f_*^{p,0})$ is a complex and the $f_q^{p,r}$ for $r \geqslant 1$ allow us to construct a morphism $C_*^p \to T_*^{p-1}$; as always, some sign precautions are needed. The construction of a multicomplex from a complex tower is not harder.

The three previous definitions can be enriched or modified according to the circumstances. For example, the complexes in a complex tower, or a bicomplex, or a multicomplex can be required to be *computable*, or *with effective homology*. These adaptations do not create any particular difficulty.

The geometric example now described is at the origin of the Serre spectral sequence. Let $\pi: E \to B$ be a simplicial fibration [MAY, Sect. 20]; then π defines a filtration on the chain complex of E; if σ is a simplex in E, its

filtration degree is p if $\pi(\sigma) \in B$ is a simplex which is the degeneracy of a *non-degenerate* p-simplex. Then C_q^p is defined as the free $\mathbb{Z}$-module generated by the $(p+q)$-simplices of E with filtration degree p, and, if $\sigma \in C_q^p$, $f_q^{p,r}(\sigma)$ is the sum of the σ faces with filtration degree $p-r$ and the right signs. It is easy to see that a multicomplex is thus well defined or, as you like, a complex tower.

8.4. DEFINITION. Let $T = (C_*^p, f^p, T_*^p)_{p \in \mathbb{N}}$ be a complex tower and $(C_q^p, f_q^{p,r})_{p,q,r \in \mathbb{N}}$ the associated multicomplex; the totalization of T is the complex T_* defined by:

(a) $T_p = \sum_{0 \leqslant r \leqslant p} C_r^{p-r}$;

(b) $f_p : C_p \to C_{p-1} = \sum_{0 \leqslant p' \leqslant p} \sum_{0 \leqslant r \leqslant p'} f_{p-p'}^{p',r}$.

This process defines what is sometimes called the *hyperhomology* of a bicomplex. Furthermore this definition allows us to interpret a complex tower as a filtration of the totalized complex.

There is only one difference from the usual definitions: here, since an effective module is free and equipped with a base, many difficulties are automatically avoided; for example, the graded complex associated to a filtration allows us to recover the original filtered complex.

We prefer to keep things simple so that we shall not give a *general spectral sequence theory*; indeed it is well known (?) that too often *general = incomprehensible*; in this section we only give a little restricted version but it is typical of what is possible to carry out in this framework. We quickly explain in the following section what is to be added in order to get the effective homology versions of Serre and Eilenberg–Moore spectral sequences; it is a matter of details wihtout any interest and which would have obscured the essential points.

8.5. THEOREM. *An algorithm can be constructed*:

input: *T, a complex tower with effective homology*;

output: *HT_*, the effective homology of T_*, the complex constructed by totalizing T.*

Therefore, the tower T not only contains the data defined in 8.1, but also the effective homology of every constituent complex.

Proof. Let HC_*^p be the computable chain complex to which the p^{th} effective complex C_*^p is homotopically equivalent. By induction on p, we construct a tower $HT = (HC_*^p, Hf^p, HT_*^p, h^p)_{p \in \mathbb{N}}$ where the additional information h^p is a homotopy equivalence between T_*^p and HT_*^p. Of course $HT_*^0 = HC_*^0$ and h^0 is one of the known homotopy equivalences. Let us assume HT is constructed up to the filtration degree $p-1$; then we have:

(a) a morphism $f^p: C_*^p \to T_*^{p-1}$;

(b) a homotopy equivalence h^{p-1} between T_*^{p-1} and HT_*^{p-1};

(c) a homotopy equivalence (given effective homology) between C_*^p and HC_*^p.

So we can apply Proposition 6.6; we obtain a morphism Hf^p: $HC_*^p \to HT_*^{p-1}$ and a homotopy equivalence h^p between the cone T_*^p of f^p and the one HT_*^p of Hf^p.

Once this work is done, by totalizing the towers T and HT, we obtain a homotopy equivalence between the effective chain complex T_* and the computable one HT_*.

In short the tower with effective homology T allows us to construct a computable tower HT and the homotopy equivalences given between chain complexes allow us to construct a homotopy equivalence between the totalized complexes of T and HT. ∎

In the usual spectral sequence process, the arrows $f^{p,r}$ can only be defined at the level $E^{p,r}$ (usually the notation E_p^r is preferred). The process described in the above theorem must be understood as a transition from $E^{*,0}$ to $E^{*,1}$; as already often explained, the fact that any appearance of torsion modules is forbidden implies that all the arrows $Hf_q^{p,r}$ can be properly "installed" between the modules $E_q^{p,1} = HC_q^p$; but the $E_q^{p,1}$ are finite-dimensional so that *it is useless continuing the process towards* $E^{*,2}, E^{*,3}, \ldots$. By simple totalization, we immediately obtain the computable complex which was desired. We shall see that in the Serre spectral sequence, two reduction steps will be needed to arrive at a tower of computable complexes $E^{p,2}$; we have preferred in this section to set aside these kinds of details so as to keep only the essential points of the described technique.

9. The Effective Homology Versions of the Serre, Eilenberg–Moore, and Adams Spectral Sequences

We only give the statements of these versions and some brief information about what is to be added to Theorem 8.5 to get a proof of them; these complements are quite inessential and have a nature well known elsewhere.

9.1. Theorem. *An algorithm can be constructed*:

input: (F, B, E, π, HF, HB) *where*:
(a) $\pi: E \to B$ *is a simplicial fibration with fibre* F *and simply connected base space* B;
(b) HF *and* HB *are the respective effective homologies of* F *and* B;

output: HE, *the effective homology of* E.

Interpretation. F, B, and E are *effective simplicial sets*, therefore ceded in a functional ways as explained in Section 4 about simplicial complexes. The simplicial morphism π and the various objects and operators describing the fiber structure are also coded in a functional way. A functor, which is easy to program, assigns to an effective simplicial set the effective associated chain complex. The data HF and HB are the effective homologies of these complexes. The same applies for the result HE.

Proof. The data allow us to construct a filtration on C_*E, the chain complex of E, by the process due to Jean-Pierre Serre. For the simplicial version, see [MAY, pp. 146–147]. As explained in Section 8, this filtration allows us to organize C_*E as a tower $\bar{T}^0$. Brown's theorem ([BRX] or [MAY, Sect. 31]) gives another tower T^0 where the p^{th} complex $C_*^p = E_*^{p,0}$ is nothing other than the tensor product $C_pB \otimes C_*F$. By applying purely and simply Theorem 8.5, this tower is homotopic to a tower T^1 whose p^{th} complex $E_*^{p,1}$ is the tensor product $C_pB \otimes H_*F$ where H_*F is the computable complex describing the effective homology of the fiber F. This description of $E_*^{*,1}$ favours the verticals. On the contrary if we favour the horizontals, we obtain a tower whose p^{th} line is the complex $C_*B \otimes H_pF$. In fact the tower notion must be generalized, for the arrows this time go from the line of index p to the lines of index $p-1$, $p+1$, $p+2$, and so on, whereas they went before from the column p to the columns $p-1$, $p-2$, ... But this does not create essential difficulties. In the same way, we construct a similar tower where the p^{th} line is now $H_*B \otimes H_pF$. But this tower is computable, so that by totalization we obtain a computable chain complex H_*E which describes the effective homology of E. ∎

We see the spectral sequence mechanism has been stopped at the level $E_*^{*,2}$ just as in Theorem 8.5 we had stopped it at the level $E_*^{*,1}$.

9.2. THEOREM. *An algorithm can be constructed*:

input: (F, B, E, π, HB, HE);

output: HF;

with the same interpretations and the same kind of hypotheses, in particular about the simple connectivity of B, *as in Theorem* 9.1.

Proof. Eilenberg and Moore proved [ELM] that the (ordinary) homology of the fiber F is a Cotor,

$$H_*F = \mathrm{Cotor}_{C_*B}(\mathbb{Z}, C_*E),$$

where the fibration projection π allows us to consider C_*E as a C_*B-comodule.

More precisely there exists a canonical homotopy equivalence between C_*F and the totalization of a bicomplex where the p^{th} column is

$$E_*^{p,0} = (\overline{C_*B})^{\otimes p} \otimes C_*E;$$

here, $\overline{C_*B}$ is the complex of B whose degree 0 component has been removed.

The vertical differentials in this bicomplex come from the boundary operators of C_*B and C_*E; the horizontal differentials defined from $E_*^{p,0}$ to $E_*^{p+1,0}$ come from the comodule structure of C_*E with respect to C_*B; the last ones are of *cobar* type. The mechanism for computing the effective homology of a bicomplex (Theorem 8.5) is then applied; there is a small difference, but a simple one: the arrows leaving the column p go to the columns $p+1$, $p+2, \ldots$ but the simple connectivity of B nevertheless ensures the convergence of the process as in a "second quadrant" spectral sequence. So we get a homotopy equivalence between $E_*^{*,0}$ and a tower $E_*^{*,1}$ where

$$E_*^{p,1} = (\overline{H_*B})^{\otimes p} \otimes H_*E;$$

We have also used the very elementary fact, but it has to be pointed out, that if we have homotopy equivalences $A \simeq A'$ and $B \simeq B'$, then we have a homotopy equivalence $A \otimes B \simeq A' \otimes B'$ so that the usual Künneth difficulties are absent.

The tower $E_*^{*,1}$ is computable, and the theorem is proved. ∎

The family of Eilenberg–Moore spectral sequences is fairly large; in addition to the one studied above, another can be used to get information about the homology of the base space of a fibration, and others work in cohomology. The method explained applies as well in these cases so as to get algorithmic techniques.

9.3. THEOREM. *Let X be an n-connected ($n \geqslant 1$) simplicial set with effective homology. Then a system (X_m, f_m^k) with $m, k \geqslant n$ can be constructed such that:*

(a) *each X_m is an m-connected simplicial set with effective homology;*

(b) *each f_m^k is a chain complex morphism $f_m^k : C_*X_m \to C_*X_{m+k}$ of degree -1;*

(c) *those f_m^k organize the $C_* X_m$ as a complex tower whose ordinary homology groups are the homotopy groups of X.*

Proof (indications only). Let us define a simplicial set X_n as

$$X_n = K(H_n(X), n) \times K(H_{n+1}(X), n+1) \times \cdots .$$

It is well known there is a canonical map from X to X_n whose homotopy fiber Y_n is "Hurewicz's error." The process can be resumed from Y_n and gives X_{n+1}, Y_{n+1}, X_{n+2}, ... If a suitable model is chosen for X_n, a canonical morphism $f_n^1 : H_* X = \pi_* X_n \to \pi_{*-1} Y_n$ (connection morphism in the Serre exact homotopy sequence) can be constructed where $H_* X$ is the *effective* homology of X, and $\pi_* Y_n$ is the "effective homotopy" of Y_n which has to be defined. But Y_n is a simplicial set with effective homology; this homology is the effective homotopy of X_{n+1} (definition) and a suitable version of the Serre exact sequence defines the effective homotopy of Y_n as the effective homology of the cone of the connection morphism $\pi_* X_{n+1} \to \pi_{*-1} Y_{n+1}$ where $\pi_{*-1} Y_{n+1}$ is the effective homotopy of Y_{n+1} which has to be defined. But Y_m is m-connected so that this definition converges. ∎

The last theorem is the effective homology version of Adams' spectral sequence.

10. Computability in Algebraic Topology

The results obtained in Sections 7 and 9 allow us to find many far-reaching computability results in algebraic topology and homological algebra. Roughly speaking, lots of computations carried out in algebraic topology can be reduced to successive applications of cone exact sequences and Serre and Eilenberg–Moore spectral sequences. With the techniques explained in this paper, these "computation" methods become actual algorithms open to theoretical (computability) and concrete applications (finding on machines homology or homotopy groups unknown until now). In this section we give some examples of applications in computability.

10.1. Theorem. *If Π is an abelian group of finite type, and if n is a positive integer, it is possible to construct a simplicial set with effective homology $K(\Pi, n)$.*

Proof. This is very easy if $n = 1$. By successive application of the effective homology version of the Eilenberg–Moore spectral sequence, the required $K(\Pi, n)$ can be constructed. ∎

Here we must point out that this proof is written essentially in the same way in Henri Cartan's famous paper [CRT] which gives the general solution for the computation problem of the homology groups of $K(\Pi, n)$'s. Indeed Cartan organizes the work as follows. To every $K(\Pi, n)$, he assigns a complex $T(\Pi, n)$, a tensor product of *elementary complexes* so that the following properties hold:

(a) the complex $T(\Pi, n)$ is *computable*; it is a finite tensor product of actually elementary, in the usual sense, complexes;

(b) if $K(\Pi, n)$ and $T(\Pi, n)$ are known, then an automatic process constructs $T(\Pi, n+1)$: this organization is *stable*; the process is of Eilenberg–Moore type and the *stability property* thus obtained leads to the direct description of $T(\Pi, n)$ with the help of *admissible sequences*;

(c) the effective simplicial set property for $K(\Pi, n)$ is obvious.

Hence Theorem 10.1 is due to Henri Cartan who actually invented effective homology theory thirty years ago.

10.2. THEOREM. *An algorithm can be constructed*:

input: (n, X) *where* X *is a simply connected simplicial set with effective homology and* n *is an integer.*

output: $\pi_n(X)$, *the* n^{th} *homotopy group of* X.

Proof. Repeatedly applying effective homology versions of Serre and Eilenberg–Moore spectral sequences, we can indeed construct on our machine the Whitehead and Postnikov towers of X and compute at the same time its Postnikov invariants. ∎

Edgar Brown [BRW] proved 10.2 for finite simplicial sets. The result obtained here gives progress in two directions. On one hand the class of simplicial sets with effective homology is much larger than the class of finite simplicial sets; a quite artificial but characteristic example was explained in the Introduction. Let us say simply here that, starting from finite simplicial sets, any "classical" construction and even any sequence of "classical" constructions (pushouts, homotopic fibers, spaces with equivariant homology, ...) always produce simplicial sets with effective homology, however, with a restriction: frequently simple connectivity (or nilpotency) properties must hold.

On the other hand, it is quite reasonable to try programming on actual machines the algorithms whose existence is thus proved. There are no specific difficulties in carrying out the explained functional programming methods, only that nothing has been done in this direction until now and

that all is yet to be done. Frequently the author is questionned about new homotopy groups of spheres. There is no good reason to be especially optimistic about this point: effective homology theory only solves *extension problems* set by the classical methods but of course cannot in any way replace the specific techniques which have been patiently worked out to go still further in the marvelous world of homotopy groups of spheres (cf. [RVN]).

On the contrary, just as Jean-Pierre Serre succeeded in computing new homotopy groups of spheres thanks to his spectral sequence, it is quite sensible to think that the methods explained here may allow us to compute on a machine the first homotopy groups of complicated spaces such as the one of the Introduction. But an important programming effort will be needed; it seems very attractive.

10.3. THEOREM. *An algorithm can be constructed*:

input: (X, n) *where* X *is a simply connected simplicial set with effective homology and* n *is an integer*;

output: *the effective homology of the null component of the* n^{th} *loop space of* X.

Proof. First we climb up the Postnikov tower in order to find X_n, the space X whose n first homotopy groups have been killed. The null component under consideration is the n^{th} loop space of X_n. Then applying n times the effective homology version of the Eilenberg–Moore spectral sequence is suffucient. ∎

Note that the cobar construction [ADM] does not solve this problem when the connectivity of X is less than $n+1$.

10.4. THEOREM. *An algorithm can be constructed*:

input: (G, K) *where* X *is a connected simplicial set with effective homology and* G *is a connected simplicial group with effective homology acting on* K;

output: $\mathrm{Bor}(G, K)$, *the space with equivariant homology of* X *with respect to the* G*-action, viewed in the form of a simplicial set with effective homology.*

Let us recall that, if $EG \to BG$ is the universal G-fibration, $\mathrm{Bor}(G, X)$ (Borel construction) is the quotient space of $X \times EG$ under the natural G-action.

Here as well the result is obtained by applying the Eilenberg–Moore spectral sequence.

A special case is important: $\mathrm{Bor}(S^1, \Omega X)$ where ΩX is the free loop space of X and where S^1 acts on the parametrization by translation. This special case occurs naturally in the study of closed geodesics on a Riemannian manifold (see [HNG]).

10.5. THEOREM. *Let G be a (discrete) perfect group whose classifying space BG is a simplicial set with effective homology. Then BG^+ is a simplicial set with effective homology.*

10.6. COROLLARY. *If the same hypotheses hold, the $K_i(\mathbb{Z}[G])$ are computable.*

Proof. Quillen's "+"-construction essentially does not change homology groups and this property holds in effective homology as well. ∎

Now let us point out a surprising but nevertheless elementary fact; it does not seem to have yet been remarked; however, its consequences should be interesting. On one hand there does not exist any general algorithm capable of deciding whether a space is simply connected; this is an easy consequence of the undecidability of the word problem in group theory. On the other hand most of the results in algebraic topology hold only when they are applied to simply connected spaces. We might as well say that from the computability point of view the situation is far from being ideal! Consequently a natural question arises: Do there exist general algorithms which *always* compute something so that, *if the input is simply connected*, something is a correct output, for example, some homotopy group? Note that you will not be able to know whether the hypothesis is satisfied but however the question makes sense! If such algorithms do exist we get a new problem: if the input does not satisfy the connectivity hypothesis, what is computed by such an algorithm? Now it is easy to see that in this way it is possible to obtain a very natural definition of Quillen's K-theory groups which does not seem to have yet been given. This point of view will be developed in detail in another paper.

REFERENCES

[ADM] J. F. ADAMS, On the cobar construction, *Proc. Nat. Acad. Sci. U.S.A.* **42** (1956), 409–412.

[AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA, 1974.

[BRW] E. H. BROWN, JR., Finite computability of Postnikov complexes, *Ann. of Math.* **65** (1957), 1–20.

[BRX] E. H. BROWN, JR., Twisted tensor products, I, *Anna. of Math.* **69** (1959), 223–246.

[CHR] A. CHURCH, A note on the Entscheidungsproblem, *J. Symbolic Logic* **1** (1936), 108–115.

[CHS] A. CHURCH, The calculi of lambda-conversion, *in* "Annals of Mathematical Studies," Vol. 6, Princeton Univ. Press, Princeton, NJ, 1941.

[CRT] H. CHARTAN, Algèbres d'Eilenberg–MacLane, *in* "Séminaire Henri Cartan, 1954–1955; Oeuvres," pp. 1309–1394, Springer-Verlag, New York/Berlin, 1979.

[DLD] A. DOLD, "Lectures on Algebraic Topology," Springer-Verlag, New York/Berlin, 1972.

[ELM] S. EILENBERG AND J. C. MOORE, Homology and fibrations. I. Coalgebras, cotensor product and its derived functors, *Comment. Math. Helv.* **40** (1966), 199–236.

[EDM] Mathematical Society of Japan, "Encyclopedic Dictionary of Mathematics," MIT Press, Cambridge, MA, 1977.

[GRR] J.-Y. GIRARD, Le λ-calcul du second ordre, *in* "Séminaire Bourbaki," Exposé 678, February 1987.

[HDG] A. HODGES, "Alan Turing the Enigma," Simon & Schuster, New York, 1983.

[HNG] N. HINGSTON, Equivariant Morse theory and closed geodesics, *J. Differential Geom.* **19** (1984), 85–116.

[HRM] H. HERMES, Enumerability, decidability, computability, *in* "Grundlehren der Mathematischen Wissenschaften," Vol. 127, Springer-Verlag, New York/Berlin, 1965.

[MAY] J. P. MAY, "Simplicial Objects in Algebraic Topology," Van Nostrand, Princeton, NY, 1967.

[MCR] J. MCCARTHY, A basis for a mathematical theory of computation, *in* "Computer Programming and Formal Systems," North-Holland, Amsterdam, 1963.

[RVN] D. C. RAVENEL, "Complex Cobordism and Stable Homotopy Groups of Spheres," Academic Press, Orlando, FL, 1986.

[SHH] W. SHIH, Homologie des espaces fibrés, *Inst. Hautes Études Sci. Publ. Math.* **13** (1962).

[SLL] D. SULLIVAN, Infinitesimal calculations in topology, *Inst. Hautes Études Sci. Publ. Math.* **47** (1977), 269–331.

[SRG] F. SERGERAERT, Homologie effective, *C.R. Acad. Sci. Paris* **304** (1987), 279–282 and 319–321.

[STL] G. STEELE, JR., "Common Lisp, the Language," Digital Press, 1984.

[TRN] A. TURING, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London. Math. Soc.* **42** (1936–1937), 230–265.