# Homologie effective des espaces de lacets itérés : un logiciel

Julio Rubio-Garcia

*THÈSE DE DOCTORAT DE MATHÉMATIQUES*
*DE L'UNIVERSITÉ JOSEPH FOURIER*

*préparée à l'Institut Fourier (Grenoble)*
*laboratoire de mathématiques*
*associé au C.N.R.S.*

# HOMOLOGIE EFFECTIVE DES ESPACES DE LACETS ITÉRÉS : UN LOGICIEL

JULIO RUBIO-GARCIA

*Soutenue à Grenoble le 25 octobre 1991 devant le jury*

*Président :* Jean-Pierre DEMAILLY
*Examinateurs :* Jean DELLA-DORA (INPG - ENSIMAG)
Michel DEMAZURE (Ecole Polytechnique)
Yves FÉLIX (Univ. Catholique Louvain-La-Neuve)
Francis SERGERAERT, Directeur de thèse
Jean-Pierre SERRE (Collège de France)

*Je tiens à remercier vivement Francis Sergeraert qui a dirigé cette thèse et sans lequel ce travail n'aurait pas vu le jour.*

*Je remercie Jean-Pierre Demailly d'avoir accepté de présider le jury et Jacques Della-Dora, Michel Demazure, Yves Félix et Jean-Pierre Serre de m'avoir fait l'honneur de faire partie de ce jury.*

*Je ne manquerai pas non plus de remercier Eladio Domínguez qui m'a fait découvrir la recherche en mathématiques et Clemens Berger et Fréderic Mouton avec lesquels j'ai partagé, à Grenoble, amitié et mathématiques.*

## SOMMAIRE

# Introduction.

L'une de nombreuses conséquences de la théorie des classes de groupes abéliens introduite par Serre [32] en 1953 est la suivante :

> Soit $X$ un espace simplement connexe. Si l'homologie de $X$ est de type fini en chaque dimension, alors il est de même pour l'homologie de son espace de lacets $\Omega X$.

Une caractéristique importante de cet énoncé est sa "stabilité" : la propriété de finitude reste stable par passage à l'espace de lacets ; ainsi le résultat admet trivialement une généralisation au cas des espaces de lacets itérés :

> Soit $X$ un espace connexe dont les $n$ premiers groupes d'homotopie sont nuls. Si l'homologie de $X$ est de type fini en chaque dimension, alors il est de même pour l'homologie de $\Omega^n X$.

Ces résultats de Serre posent le problème du calcul de l'homologie des espaces de lacets itérés. Pourtant les démonstrations de Serre reposent fortement sur la suite spectrale qui porte son nom et le calcul direct de l'homologie par cette méthode reste difficile dans le cas général (problèmes d'extension).

Ces théorèmes assurent d'ailleurs *l'existence* de complexes de chaînes de type fini en chaque dimension et homotopiquement équivalents aux complexes singuliers des espaces de lacets. Il était donc naturel de chercher des méthodes directes de *construction* de tels complexes de type fini.

Un premier essai fut accompli par Adams et Hilton [2] en 1956 qui associent à un CW-complexe $X$, fini et à 1-squelette réduit à un point, un complexe de chaînes libre de type fini en chaque dimension dont l'homologie est isomorphe à l'homologie de $\Omega X$.

Cette construction manquait de stabilité (passage d'un CW-complexe à un complexe de chaînes) et pour éviter cette difficulté Adams introduit sa célèbre *construction Cobar*. Dans l'introduction de [1], Adams indique que l'un des buts recherchés est la possibilité d'itérer le procédé, car on y définit le passage du complexe de chaînes d'un CW-complexe à un autre complexe de chaînes (la construction Cobar). Mais en fait la construction Cobar n'est pas stable : on y utilise de façon essentielle la structure de coalgèbre canoniquement portée par le complexe de chaînes de départ. Le problème de

l'itération de la construction Cobar (autrement dit, le problème consistant à définir un coproduit sur la construction Cobar compatible avec celui de l'espace de lacets) est resté ouvert jusqu'à 1980 quand Baues [6] (voir aussi [7]) a démontré que la construction Cobar peut être itérée une fois (à condition que l'espace de départ soit à 2-squelette trivial), mais on ne peut cette fois trouver un "bon" coproduit sur le deuxième Cobar, ce qui empêche de calculer par cette méthode l'homologie des espaces de lacets itérés plus de deux fois.

Dans ce travail on réétudie le problème de trouver un complexe de chaînes de type fini en chaque dimension ayant la même homologie qu'un espace de lacets itéré. Le point clé dans la démonstration de Serre sur la finitude de l'homologie des espaces de lacets est un théorème très général reliant l'homologie des trois composantes dans une fibration. Ensuite on tire parti de l'existence du fibré associé à un espace de lacets, d'espace total contractile, pour en déduire des propriétés sur les espaces de lacets.

Ce schéma est aussi suivi dans notre travail. En utilisant les techniques de l'homologie effective (voir [31]), on modifie convenablement la suite spectrale d'Eilenberg-Moore [14] pour en obtenir un algorithme dont les entrées sont les homologies effectives de la base et l'espace total d'un fibré et la sortie est l'homologie effective de la fibre. En particularisant ensuite ce résultat au cas des espaces de lacets, on atteint le but cherché :

> Soit $X$ un ensemble simplicial à 1-squelette trivial. Si $X$ est à homologie effective, alors son espace (simplicial) de lacets est à homologie effective.

Autrement dit, on décrit un algorithme construisant un complexe de chaînes de type fini et une équivalence d'homotopie explicite entre ce complexe et celui de l'espace de lacets. Puisque cet énoncé est clairement "stable", la solution du cas des espaces de lacets itérés $n$ fois est un simple corollaire (en ajoutant la condition de trivialité sur le $n$-squelette).

Cet algorithme fournit en même temps une solution complète au problème d'Adams sur l'itération de la construction Cobar. Le complexe de type fini construit peut être interprété comme une construction Cobar "homotopique" d'une coalgèbre à homotopie près (le coproduit n'étant associatif qu'à homotopie près). Ces notions avaient été déjà utilisées par Stasheff [34], qui,

apparemment, n'a pas étudié le problème de l'itération de ces constructions. Voir aussi [23] où Lambe et Stasheff appliquent des méthodes comparables dans une situation itérative différente.

L'algorithme évoqué plus haut a été implanté sur machine sous forme d'un programme Lisp. Dans ce programme on utilise de façon essentielle la technique du codage fonctionnel (voir [31]) qui permet de tourner les difficultés posées par le traitement des ensembles infinis, notamment des espaces de lacets.

Dans ce logiciel la question de la stabilité mentionnée plusieurs fois dans cette introduction se traduit par le fait de qu'une *seule* fonction Lisp sert à calculer l'homologie effective de n'importe quel espace de lacets, étant supposé que l'argument est un ensemble simplicial à homologie effective et à 1-squelette trivial. L'itération banale de cette fonction Lisp résout *théoriquement* le problème pour les espaces de lacets itérés $n$ fois. Évidemment la complexité des algorithmes fait que le choix de $n$ soit assez restreint, mais on a déjà calculé des groupes d'homologie d'espaces de lacets itérés deux ou trois fois qui semblaient jusqu'à présent hors de portée des méthodes de calcul connues.

Ce mémoire est organisé en 5 chapitres. Dans le premier chapitre, on définit les bases théoriques permettant d'énoncer correctement les résultats de calculabilité démontrés dans les chapitres suivants. Dans la première section, on introduit un modèle théorique de calcul (inspiré du $\lambda$-calcul). La deuxième section est dédiée aux notions de codage fonctionnel et relationnel et dans la troisième on particularise cette sorte de codages aux objets de l'algèbre homologique.

Le chapitre 2 traite de l'algèbre homologique élémentaire. Ses trois sections développent les théories de l'homologie effective et de la perturbation homologique et leur application pour le calcul de l'homologie des bicomplexes.

Une étude détaillée des théorèmes d'Eilenberg-Zilber et de Brown-Shih (Eilenberg-Zilber "tordu") est entreprise dans le troisième chapitre. Dans la section 3 de ce chapitre les similitudes et les différences entre le théorème d'Eilenberg-Zilber tordu et le théorème de Brown sur l'existence des cochaînes de torsion sont systématiquement étudiées.

Le chapitre 4 est la partie essentielle de ce travail : la suite spectrale d'Eilenberg-Moore y est transformée en un *vrai* outil de calcul et on y décrit un algorithme permettant de calculer l'homologie des espaces de lacets itérés. Ce chapitre est une version avec démonstrations détaillées de l'article [29].

Enfin, le dernier chapitre traite du logiciel implémentant les algorithmes décrits dans les autres chapitres. Dans la première section on compare les codages concrets utilisés aux codages théoriques du chapitre 1 et on évoque brièvement les problèmes de complexité posés par notre logiciel. La relation entre les calculs sur machine et les résultats obtenus par application des méthodes connues auparavant est étudiée dans la section suivante. La dernière section est constituée par une liste des groupes d'homologie calculés à l'aide du logiciel.

# 1 Machines et codages.

## 1.1 Machine.

**Définition 1.1.1** *Une* machine *(théorique) est un quadruplet* $(\mathcal{E},\mathcal{U},\rho,\alpha)$ *où :*

1. $\mathcal{E}$ *est un ensemble dénombrable,* $\mathcal{U}$ *est un sous-ensemble de* $\mathcal{E}$ *;*
2. $\rho$ *est une application* $\rho : \mathcal{E} - \mathcal{U} \to \mathcal{E}$ *;*
3. $\alpha$ *est une application* $\alpha : \mathcal{U} \times \mathcal{U} \to \mathcal{E}$.

**Interprétation intuitive.**

- $\mathcal{E}$ est l'ensemble des états de la machine.
- $\mathcal{U}$ (pour *univers*) est le sous-ensemble des états terminaux de la machine.
- $\rho$ décrit une étape (élémentaire) de calcul.
- $\alpha$ décrit le mécanisme de "mise en marche" de la machine.

A toute machine théorique $(\mathcal{E},\mathcal{U},\rho,\alpha)$ est associé un *modèle de calcul* $(\mathcal{U},\gamma,?)$ où l'application $\gamma : \mathcal{U} \times \mathcal{U} \to \mathcal{U} \cup \{?\}$ (? est un symbole non élément de $\mathcal{E}$) est défini comme suit :

soient $t_1, t_2 \in \mathcal{U}$ ; on pose $x_0 := \alpha(t_1, t_2) \in \mathcal{E}$ et on définit par récurrence $x_n := \rho(x_{n-1})$, lorsque $x_{n-1} \notin \mathcal{U}$. Deux cas sont alors possibles :

1. il existe un entier $n$ tel que $x_n \in \mathcal{U}$ ; alors $\gamma(t_1, t_2) := x_n$.
2. l'élément $x_n$ n'est jamais élément de $\mathcal{U}$ ; il est donc défini pour tout $n$ et on pose alors $\gamma(t_1, t_2) :=?$.

**Interprétation intuitive.** L'état initial $x_0$ est l'état à partir duquel le programme $t_1$ s'apprête à "travailler" sur $t_2$. Par ailleurs $\gamma$ est la fonction définissant un calcul "complet" à partir des données $t_1, t_2$. Si le calcul ne termine pas, on définit $\gamma(t_1, t_2)$ comme le symbole spécial ?. En général, l'utilisateur ne peut deviner d'avance si le calcul va ou ne va pas terminer.

On peut en suivant ce schéma, définir une machine théorique à partir du $\lambda$-calcul. La caractéristique principale de cette machine sera que les éléments de $\mathcal{U}$ auront une interprétation "fonctionnelle".

Etant donné un ensemble $\mathcal{A} = \{a, b, c, \ldots, f, g, \ldots, x, \ldots\}$ (un *alphabet*), infini dénombrable, et les symboles "?","$\lambda$","[","]" et "." (supposés non éléments de $\mathcal{A}$), on peut définir l'ensemble des *termes du $\lambda$-calcul* (sur $\mathcal{A}$), récursivement, comme suit :

1. à chaque terme $\tau$ élément de cet ensemble, on associe trois sous-ensembles de $\mathcal{A}$, nommés *lettres*($\tau$), *libres*($\tau$) et *liées*($\tau$) vérifiant *lettres*($\tau$) = *libres*($\tau$) $\cup$ *liées*($\tau$) et *libres*($\tau$) $\cap$ *liées*($\tau$) = $\emptyset$. Avec cette propriété, en définissant *libres*($\tau$) et *liées*($\tau$), *lettres*($\tau$) est complètement déterminé.

2. chaque terme est construit par l'un des trois procédés suivants :

   - les éléments $\kappa$ de $\mathcal{A}$ sont des termes ; *libres*($\kappa$) = $\{\kappa\}$ et *liées*($\kappa$) = $\emptyset$.
   - si $\kappa$ est un élément quelconque de $\mathcal{A}$ et $\tau$ représente un terme vérifiant $\kappa \in$ *libres*($\tau$), alors $\lambda\kappa.\tau$ est un terme, *libres*($\lambda\kappa.\tau$) = *libres*($\tau$) $-$ $\{\kappa\}$ et *liées*($\lambda\kappa.\tau$) = *liées*($\tau$) $\cup$ $\{\kappa\}$.
   - si $\tau_1$, $\tau_2$ sont deux termes vérifiant *libres*($\tau_1$) $\cap$ *liées*($\tau_2$) = *libres*($\tau_2$) $\cap$ *liées*($\tau_1$) = $\emptyset$, alors $[\tau_1\ \tau_2]$ est un terme, *libres*($[\tau_1\ \tau_2]$) = *libres*($\tau_1$)$\cup$*libres*($\tau_2$) et *liées*($[\tau_1\ \tau_2]$) = *liées*($\tau_1$)$\cup$ *liées*($\tau_2$).

L'ensemble des termes va être l'ensemble des états de notre machine théorique et on le notera donc $\mathcal{E}$. Les objets *lettres*, *libres* et *liées* deviennent trois fonctions de $\mathcal{E}$ vers $\mathcal{P}(\mathcal{A})$. Remarquer que *lettres*($\tau$), où $\tau$ est un terme, coïncide avec l'ensemble des éléments de $\mathcal{A}$ apparaissant dans l'écriture de $\tau$.

**Interprétation intuitive.**

- une lettre (un élément de $\mathcal{A}$) est à considérer comme une "fonction abstraite" sur laquelle on ne dispose d'aucune information particulière.

- $\lambda\kappa.\tau$ est à considérer comme une fonction dont l'argument est $\kappa$ et la tâche à réaliser est définie par $\tau$.
- dans $\lambda\kappa.\tau$, $\kappa$ est une lettre liée ; elle y joue le rôle de variable muette (paramètre de définition de fonction en termes des langages de programmation).
- $[\tau_1\ \tau_2]$ est l'état initial d'une machine où on s'apprête à faire travailler la fonction $\tau_1$ sur l'argument $\tau_2$.
- enfin, les lettres libres dans un terme jouent le même rôle que les variables externes à une procédure dans les langages de programmation.

**Définition 1.1.2** *Un terme du $\lambda$-calcul est* en forme normale *si dans son écriture ne figure pas la sous-chaîne "$[\lambda$".*

Définissons maintenant notre machine théorique (en fait une approximation ; voir les remarques qui suivent) à partir des termes du $\lambda$-calcul :

- $\mathcal{E} = \{$termes du $\lambda$-calcul (sur $\mathcal{A})\}$.
- $\mathcal{U} = \{$termes en forme normale$\}$.
- $\rho : \mathcal{E} - \mathcal{U} \to \mathcal{E}$. Soit $\tau \in \mathcal{E} - \mathcal{U}$ ; alors il y a des sous-chaînes "$[\lambda$" dans l'écriture de $\tau$. Soit $[\lambda\kappa.\tau_1\ \tau_2]$ le *sous-terme* de $\tau$ commençant par la sous-chaîne"$[\lambda$" située le plus à gauche et supposons que $\tau$ s'écrive $ch_1[\lambda\kappa.\tau_1\ \tau_2]ch_2$. On définit alors $\rho(\tau) := ch_1\tau' ch_2$ où $\tau'$ est le terme obtenu à partir de $\tau_1$ en remplaçant chaque ocurrence de $\kappa$ dans $\tau_1$ par $\tau_2$.
- Etant donné $\tau_1, \tau_2 \in \mathcal{U}$, on définit $\alpha(\tau_1, \tau_2) := [\tau_1\ \tau_2]$.

Désormais nous fixons ainsi cette machine théorique $(\mathcal{E}, \mathcal{U}, \rho, \alpha)$ et le modèle de calcul $(\mathcal{U}, \gamma, ?)$ correspondant.

**Remarques.**

1. Cette définition de $\mathcal{E}$ est approximative. Ce qu'on devrait définir comme ensemble des états n'est pas l'ensemble des termes du $\lambda$-calcul,

mais plutôt un quotient de cet ensemble. La relation d'équivalence consiste à identifier certains termes ne différant que par les noms de certaines de leurs lettres liées. La définition rigoureuse de cette relation se trouve dans [28], chapitre 2, pp. 19-20, où elle est nommée $\alpha$-équivalence.

Cette relation explicite l'interprétation de $\kappa$ comme une variable muette dans $\lambda\kappa.\tau$. Par exemple, le terme $\lambda x.[f\ x]$ est $\alpha$-équivalent à $\lambda y.[f\ y]$. De même, $\lambda f.\lambda x.[f\ x]$ est $\alpha$-équivalent à $\lambda g.\lambda y.[g\ y]$. Par contre, $\lambda x.[f\ x]$ n'est pas $\alpha$-équivalent à $\lambda y.[g\ y]$.

2. De même, la définition donnée pour $\rho$ est incomplète. Des interférences éventuelles entre variables doivent être considérées, rendant la définition de $\rho$ un peu plus complexe (voir la notion de $\beta$-équivalence dans [28], chapitre 2).

3. L'étape élémentaire de calcul $\rho$ est à comprendre comme une *règle de réécriture* ou encore comme une *évaluation* de la fonction $\lambda\kappa.\tau_1$ lorsque le "paramètre" $\kappa$ prend la valeur $\tau_2$.

**Définition 1.1.3** *Un terme $\tau$ est un* combinateur *(combinator en anglais) si $libres(\tau) = \emptyset$. Deux combinateurs sont égaux s'ils sont $\alpha$-équivalents (voir la remarque 1 précédente).*

**Exemples.**
Les termes suivants sont des combinateurs :

- $\lambda f.\lambda x.[f\ x]$
- $\lambda f.\lambda x.[f\ [f\ x]]$

En général, le *$n$-ième entier de Church*, noté $c_n$ ($n$ entier positif), est le combinateur $\lambda f.\lambda x.[f\ [f \ldots [f\ x]] \ldots]$, où dans $[f\ [f \ldots [f\ x]] \ldots]$ figurent $n$ occurrences de la lettre $f$.

L'intérêt des combinateurs est qu'ils sont "indépendants de l'environnement" et ont donc une sémantique fixée. C'est le cas des entiers de Church qui représentent en $\lambda$-calcul les nombres entiers (positifs) ordinaires, autrement dit les éléments de $\mathbf{N}$.

Le $\lambda$-calcul ne permet donc de travailler qu'avec des "variables" (lettres), des expressions de fonctions et des applications de fonctions. Mais, compte-tenu du commentaire précédent, il est naturel de remplacer les combinateurs par des abréviations (des symboles) afin de rendre moins lourde l'écriture des termes. Ceci est essentiellement l'idée de McCarthy, le créateur de Lisp, de sorte que, très schématiquement :

$$\lambda\text{-calcul} \quad + \quad \text{symboles} \quad = \quad \text{Lisp}$$

**Exemple.**
Supposons qu'on ait choisi $*$ comme abréviation du combinateur

$$\lambda m.\lambda n.\lambda f.[m\ [n\ f]]$$

et que 2, 3 notent les entiers de Church $c_2$ et $c_3$, respectivement.

Le lecteur peut deviner la sémantique du combinateur $*$ en évaluant

$$[[*\ 2]\ 3]\ ;$$

par application des règles de réécriture expliquées précédemment, il trouvera $\gamma(\gamma(*,2),3) = 6$.

Les opérations arithmétiques usuelles peuvent ainsi être "programmées" sur notre machine à $\lambda$-calcul.

De plus, à partir du $\lambda$-calcul pur et d'un symbole réservé, nommé traditionnellement $NIL$, il est possible (voir [28], chapitre 4) d'introduire des termes du $\lambda$-calcul représentant les listes et des combinateurs permettant les opérations usuelles sur les listes. Le symbole $NIL$ représente la liste vide. Ainsi la liste dont les éléments sont les combinateurs 1, 2, 3 (ce qu'en langage Lisp on écrit `(1 2 3)`) peut être représenté en $\lambda$-calcul par le terme

$$\lambda a.[[a\ 1]\ \lambda b.[[b\ 2]\ \lambda c.[[c\ 3]\ NIL]]].$$

Pour donner un exemple, considérons l'opération d'extraction de la "queue" d'une liste. En terminologie Lisp, cela correspond à l'opérateur `cdr` ; ainsi `cdr` travaillant sur `(1 2 3)` donne la liste `(2 3)`. Le combinateur

$$\lambda x.[x\ \lambda u.\lambda v.v]$$

correspond à la fonction Lisp `cdr`, comme le lecteur peut vérifier en calculant

$$\gamma(\lambda x.[x\ \lambda u.\lambda v.v], \lambda a.[[a\ 1]\ \lambda b.[[b\ 2]\ \lambda c.[[c\ 3]\ NIL]]]),$$

ce qui donne bien le terme $\lambda b.[[b\ 2]\ \lambda c.[[c\ 3]\ NIL]]$ (la liste `(2 3)`).

Comme il est bien connu, les symboles et les listes sont les ingrédients de base du langage Lisp ; par ailleurs l'algorithme d'évaluation (autrement dit, d'exécution des programmes) des machines Lisp est directement inspiré par le mécanisme de réécriture expliqué plus haut. Il n'est pas donc difficile d'écrire un "compilateur" traduisant les expressions Lisp en termes du $\lambda$-calcul. Cela nous permettra d'utiliser dans la suite une notation Lisp lorsqu'il s'agit de décrire des algorithmes implantés sur notre machine théorique.

Par exemple, la traduction de l'expression Lisp

```
((lambda (a) (* a a)) 2)
```

est le terme

$$[\lambda a.[[\lambda m.\lambda n.\lambda f.[m\ [n\ f]]\ a]\ a]\ 2].$$

Un autre exemple déjà cité : la traduction de

```
(cdr (1 2 3))
```

est le terme

$$[\lambda x.[x\ \lambda u.\lambda v.v]\ \lambda a.[[a\ 1]\ \lambda b.[[b\ 2]\ \lambda c.[[c\ 3]\ NIL]]]].$$

Les listes interviennent dans les expressions Lisp au moins de trois façons différentes :

1. D'abord, elles servent à *programmer* les "exécutions" de fonctions, ce qui, en $\lambda$-calcul, se fait à l'aide de l'opérateur binaire [. .] ; ainsi la liste Lisp `(f x)` correspond plus ou moins à l'expression $[f\ x]$ du $\lambda$-calcul.

2. Elles servent à *définir* des fonctions, presque toujours des "combinateurs", autrement dit des fonctions sans variable externe ; la liste Lisp `(lambda (x) ...)` correspond ainsi à l'expression $\lambda x \ldots$ du $\lambda$-calcul.

3. Enfin, elles sont employées comme structures de données. Comme il a été déjà remarqué, la traduction de listes telle que `(1 2 3)` en $\lambda$-calcul (étendu à l'aide du symbole $NIL$) se fait sans problème majeur (voir [28], chapitre 4).

Ce traducteur respecte les règles d'évaluation Lisp et $\lambda$-calcul. Par exemple, l'évaluation de `((lambda (a) (* a a)) 2)` produit le nombre 4 et la réécriture du terme correspondant donne bien le combinateur qu'on a aussi nommé 4.

Les fonctions à plusieurs arguments sont traitées en $\lambda$-calcul comme des applications enchaînées de fonctions. Par exemple, l'expression Lisp `(* a a)` est traduite comme $[[*\ a]\ a]$.

Noter enfin qu'une fois admise l'utilisation des symboles et des listes, il n'est pas difficile d'implanter les structures (RECORDS en langages type ALGOL). On employera aussi dans la suite ce type d'objets.

## 1.2 Codage.

On veut associer aux objets de la topologie algébrique des modèles sur une machine (théorique) de façon à énoncer et démontrer des résultats de calculabilité. On fixe comme machine le quadruplet $(\mathcal{E},\mathcal{U},\rho,\alpha)$ défini dans le paragraphe précédent à partir du $\lambda$-calcul ; on utilisera aussi le modèle de calcul $(\mathcal{U},\gamma,?)$ associé. Compte-tenu des remarques finales du paragraphe 1, on utilisera des notations Lisp pour représenter les termes du $\lambda$-calcul.

Etant donné une classe d'objets (mathématiques), on lui associera dans les bons cas un sous-ensemble de $\mathcal{U}$ dont les éléments représenteront (dans un sens qu'il faudra préciser) les objets mathématiques de cette classe.

Définissons une *$\mathcal{U}$-classe* comme un sous-ensemble de $\mathcal{U}$. Notons alors $\mathcal{U}-CLASSES$ comme l'ensemble des $\mathcal{U}$-classes ; autrement dit, $\mathcal{U}-CLASSES = \mathcal{P}(\mathcal{U})$.

On note **Bool** $= \{true, false\}$ la $\mathcal{U}$-classe à deux éléments destinés à être les résultats des calculs de prédicats, et $\mathcal{N}_{\mathcal{U}}$ (ou simplement $\mathcal{N}$ si aucune confusion n'est à craindre) la $\mathcal{U}$-classe des entiers de Church (voir [28] pp. 38 et pp. 40, respectivement).

On peut définir une loi de composition sur les $\mathcal{U}$-classes par l'opérateur

$$\mathcal{U}-CLASSES \times \mathcal{U}-CLASSES \xrightarrow{\Phi} \mathcal{U}-CLASSES$$

défini par : $\Phi(\mathcal{A}_1,\mathcal{A}_2) = \{f \in \mathcal{U};\ \forall t \in \mathcal{A}_1, \gamma(f,t) \in \mathcal{A}_2\}$.

L'interprétation intuitive de $\Phi(\mathcal{A}_1,\mathcal{A}_2)$ est la suivante : c'est l'ensemble des programmes qui, s'ils prennent comme argument une donnée de la $\mathcal{U}$-classe $\mathcal{A}_1$, produisent un résultat de la $\mathcal{U}$-classe $\mathcal{A}_2$.

On a aussi les constructeurs de $\mathcal{U}$-classes correspondant aux listes : étant donné $n$ $\mathcal{U}$-classes $\mathcal{A}_1,\ldots,\mathcal{A}_i,\ldots,\mathcal{A}_n$, on considère la $\mathcal{U}$-classe $(\mathcal{A}_1,\ldots,\mathcal{A}_i,\ldots,\mathcal{A}_n)$ dont les éléments sont les listes de longueur $n$ telles que le $i$-ème élément appartient à $\mathcal{A}_i$ $(i = 1,\ldots,n)$. On a de même les constructeurs de $\mathcal{U}$-classes associés aux structures (voir la fin du paragraphe précédent).

Pour une première approche de la notion de codage, considérons à titre d'exemple le cas où la classe des objets à coder est l'ensemble des nombres entiers positifs **N**. Bien entendu, le candidat naturel pour coder un nombre est l'entier de Church correspondant. Soit $\mathcal{N}$ la $\mathcal{U}$-classe dont les éléments sont les entiers de Church. Compte-tenu de la convention fixée dans le paragraphe

précédent de noter les entiers de Church 0, 1, 2, 3, ..., cet exemple de codage peut sembler un peu trivial : la bijection canonique entre $\mathcal{N}$ et $\mathbf{N}$ ressemble à l'application identité !

L'application $\mathcal{N} \to \mathbf{N}$ est nommée *application de décodage* et l'application $\mathbf{N} \to \mathcal{N}$ *application de codage*.

En général, un *codage* d'une classe d'objets $C$ est un couple $(\mathcal{C}, \chi_C)$ où $\mathcal{C}$ est une $\mathcal{U}$-classe et $\chi_C : \mathcal{C} \to C$ est une application (l'application de *décodage*). Un inverse à droite de $\chi_C$ (s'il en existe un) est une *application de codage*.

Un codage est *complet* si l'application de décodage est surjective (autrement dit, s'il existe une application de codage, en général non unique), *univoque* si elle est injective et *parfait* si elle est bijective.

Le codage de $\mathbf{N}$ donné plus haut est évidemment parfait. Des codages non parfaits interviennent naturellement dans beaucoup de questions. Par exemple, étant donné l'ensemble $\mathbf{N}^{\mathbf{N}}$ des fonctions de $\mathbf{N}$ vers $\mathbf{N}$, il est naturel de choisir comme $\mathcal{U}$-classe $\Phi(\mathcal{N},\mathcal{N})$ avec l'application de décodage évidente. Cette application ne peut pas être surjective (car $\mathbf{N}^{\mathbf{N}}$ est non dénombrable alors que $\Phi(\mathcal{N},\mathcal{N}) \subset \mathcal{U}$ est dénombrable) et n'est pas non plus injective (deux programmes différents peuvent définir la même fonction).

Ce dernier exemple est un codage dit *fonctionnel*. Soient $E$, $F$, $G$ trois ensembles et $\chi_E : \mathcal{E} \to E$ et $\chi_F : \mathcal{F} \to F$ deux codages ; soit enfin $\beta : F^E \to G$ une application quelconque, où $F^E$ est l'ensemble des applications de $E$ vers $F$. Alors un *codage fonctionnel* pour $G$ est défini en choisissant la $\mathcal{U}$-classe

$$\mathcal{G} = \{\tau \in \Phi(\mathcal{E},\mathcal{F}) \,;\, si\ e, e' \in \mathcal{E}\ et\ \chi_E(e) = \chi_E(e')\ alors\ \gamma(\tau, e) = \gamma(\tau, e')\}$$

et l'application de décodage $\chi_G : \mathcal{G} \to F^E \xrightarrow{\beta} G$ évidente.

Décrivons un autre exemple de codage fonctionnel. Soient $G = \mathcal{P}(\mathbf{N})$, $E = \mathbf{N}$, $F = \mathbf{Bool}$ (le sous-ensemble de $\mathcal{U}$ dont les éléments sont $true$, $false$) ; $\mathcal{E} = \mathcal{N}$, $\mathcal{F} = F$ avec les codages parfaits évidents. Alors $F^E$ peut être naturellement interprété comme l'ensemble des fonctions caractéristiques des éléments de $\mathcal{P}(\mathbf{N})$ et on peut prendre pour $\beta : F^E \to G$ la bijection habituelle. Ces données suffisent pour définir un *codage fonctionnel* de $\mathcal{P}(\mathbf{N})$, où la $\mathcal{U}$-classe choisie est $\Phi(\mathcal{N}, \mathbf{Bool})$ (car $\chi_E$ est dans ce cas une bijection). Ce codage n'est ni univoque ni complet.

Comme le dernier exemple le montre, la technique du codage fonctionnel permet le codage d'objets infinis, bien que certaines informations sur l'objet codé sont en fait fréquemment inaccessibles. Par exemple, il n'existe pas d'algorithme capable de déterminer à partir d'un élément de $\Phi(\mathcal{N}, \mathbf{Bool})$ si le sous-ensemble de $\mathbf{N}$ codé est vide ou non.

Cette incomplétitude de l'information sur les ensembles codés n'interdit pourtant pas d'effectuer des calculs intéressants à partir de ces codes. En particulier, les constructions habituelles sur les ensembles (complémentation, réunion, intersection, ...) peuvent être "programmées" dans ce cadre. Précisons ce qu'on entend par "programmées".

D'abord, quelques notations. Soient $\mathcal{A}_1$, $\mathcal{A}_2$ deux $\mathcal{U}$-classes et soit $\tau$ un élément de $\Phi(\mathcal{A}_1, \mathcal{A}_2)$ ; on introduit alors une application $\Gamma_\tau : \mathcal{A}_1 \to \mathcal{A}_2$ définie par : $\Gamma_\tau(a_1) := \gamma(\tau, a_1)$. Ainsi $\Gamma_\tau$ n'est autre que la réinterprétation de l'algorithme $\tau$ comme une application ensembliste entre ses entrées et ses sorties.

**Définition 1.2.1** *Soient $C_1$, $C_2$ deux classes d'objets mathématiques, $F : C_1 \to C_2$ une certaine "construction" mathématique (par exemple, la partie d'un foncteur qui agit sur les objets d'une catégorie), et soient enfin $\chi_1 : C_1 \to C_1$, $\chi_2 : C_2 \to C_2$ deux décodages. Affirmer la* calculabilité *de $F$, c'est montrer l'existence d'un élément $\tau \in \Phi(C_1, C_2)$ vérifiant : $F\chi_1 = \chi_2\Gamma_\tau$ ;* calculer *$F$ consiste à* produire *un tel élément $\tau$.*

Cette définition s'étend d'une façon naturelle aux constructions "à plusieurs arguments".

Par exemple, l'opération de complémentation sur les sous-ensembles de $\mathbf{N}$ est calculable par rapport au codage fonctionnel donné plus haut. Un terme du $\lambda$-calcul (écrit en notation Lisp) qui code cette construction est :

```
(lambda (ensemble)
   (lambda (n)
      (not (ensemble n))))
```

De même, la réunion et l'intersection sont codées à partir des opérateurs logiques `or` et `and`, respectivement.

Malgré l'efficacité de cette notion de codage (qui permet notamment de travailler sur machine avec des ensembles infinis), il y a des situations

courantes en programmation (théorique ou concrète) qui ne peuvent être modélisées dans ce cadre. Considérons une famille $C$ de fonctions, un anneau de fonctions différentiables à une variable, par exemple. Il est tout à fait usuel en calcul numérique d'utiliser, pour "simuler" un tel anneau, une autre famille contenant des versions "discrètes" des fonctions. Etant donné $n$ valeurs $x_1, \ldots, x_n$ de la variable, on code une fonction par un polynôme qui coïncide en $x_1, \ldots, x_n$ avec la fonction donnée. Dans une telle situation, le code sur machine d'une fonction de $C$ peut être une liste de nombres (les coefficients du polynôme). Pourtant, on ne peut déterminer entièrement la fonction à partir d'un tel code. On se trouve donc dans une situation où une "technique de codage", largement utilisée pour résoudre de vrais problèmes de calcul, ne s'adapte pas à la définition de codage donnée précédemment. D'où la nécessité d'introduire une notion de codage plus générale.

**Définition 1.2.2** *Etant donné une classe d'objets mathématiques $C$, un* codage relationnel *de $C$ est défini par une relation entre une $\mathcal{U}$-classe $\mathcal{C}$ et $C$ ; autrement dit, un codage relationnel n'est qu'un sous-ensemble $\mathcal{R}$ du produit cartesian $\mathcal{C} \times C$. Les applications de codage et de décodage sont alors définies comme la première et la deuxième projections canoniques de $\mathcal{R}$.*

Dans l'exemple du codage des fonctions différentiables, supposons, afin de simplifier les notations, que les $n$ valeurs $x_1, \ldots, x_n$ de la variable sont fixées pour toutes les fonctions de la famille $C$. On sait alors que les polynômes d'interpolation sont au plus de degré $n-1$ et la liste des coefficients est donc de longueur $n$. On peut ainsi choisir comme $\mathcal{U}$-classe $\mathcal{C}$ la réunion des listes de longueur $n$ dont les éléments sont des nombres. Si on considère qu'une liste $p$ de nombres représente les coefficients d'un polynôme, étant donné un nombre $x$, on peut associer à $p$ et à $x$ la valeur au point $x$ du polynôme représenté par $p$ ; notons $p(x)$ cette valeur. Avec cette notation, la relation cherchée pour définir le codage n'est autre que

$$\mathcal{R} = \{(p, f) \; ; \; p(x_i) = f(x_i), i = 1, \ldots, n\}.$$

La nouvelle notion généralise celle qui a été définie précédemment : étant donné un codage $(\mathcal{C}, \chi_C)$ pour une classe d'objets $C$, le codage relationnel correspondant n'est autre que

$$\mathcal{R} = \{(x, y) \in \mathcal{C} \times C \; ; \; \chi_C(x) = y\}.$$

**Définition 1.2.3** *Si $\mathcal{R}_1 \subset \mathcal{C}_1 \times C_1$, $\mathcal{R}_2 \subset \mathcal{C}_2 \times C_2$ sont deux codages relationnels et si $F : C_1 \to C_2$ est une construction mathématique, alors la construction $F$ est* calculable *par rapport aux codages relationnels $\mathcal{R}_1$ et $\mathcal{R}_2$ s'il existe un terme $\tau \in \Phi(\mathcal{C}_1, \mathcal{C}_2)$ vérifiant la condition : l'image de l'application produit $\Gamma_\tau \times F$ (dont la source est $\mathcal{R}_1$) est contenue dans $\mathcal{R}_2$ ; autrement dit, on a : $\Gamma_\tau \times F : \mathcal{R}_1 \to \mathcal{R}_2$.*

Comme dans le cas du codage "traditionnel", cette définition peut être aisément généralisée pour les constructions mathématiques "à plusieurs arguments".

Dans l'exemple du codage relationnel des fonctions différentiables, une construction calculable est l'addition des fonctions. Puisque les $n$ points $x_1, \ldots, x_n$ sont communs pour toutes les fonctions codées et que les codes sont toujours des listes de la même longueur $n$, le terme $\tau$ correspondant s'écrit en (pseudo) Lisp :

```
(lambda (liste1 liste2)
   (mapcar + liste1 liste2))
```

où `mapcar` est le combinateur qui applique une fonction (la fonction `+` dans ce cas) aux éléments correspondants des listes arguments et qui retourne une liste des résultats.

## 1.3 Codage en algèbre homologique.

Dans ce paragraphe on va particulariser notre schéma de codage aux objets de l'algèbre homologique. On décrit en détail le cas des groupes abéliens libres, on traite brièvement le codage des complexes de chaînes et on laisse au lecteur imaginer le codage des objets plus complexes.

Dans la suite, nos groupes abéliens libres seront toujours *à base distinguée.* Plus précisement une telle base sera toujours le quotient d'une $\mathcal{U}$-classe par une relation d'équivalence. On traite donc d'abord le problème du codage des quotients de $\mathcal{U}$-classes.

### 1.3.1 Quotients de $\mathcal{U}$-classes.

On se restreint à étudier les quotients tels que les $\mathcal{U}$-classes et les relations d'équivalence définissant ces quotients vérifient deux propriétés :

1. Une $\mathcal{U}$-classe à partir de laquelle on construit un quotient doit être munie d'une fonction caractéristique exprimable comme un terme du $\lambda$-calcul. Autrement dit, pour une telle $\mathcal{U}$-classe $\mathcal{B}$, il existe $\mathtt{b} \in \Phi(\mathcal{U}, \mathbf{Bool})$ vérifiant :

$$\mathcal{B} = \{\tau \in \mathcal{U} \ ; \ \Gamma_{\mathtt{b}}(\tau) = \mathit{true}\}.$$

   Par exemple, la $\mathcal{U}$-classe $\mathcal{Z}$ qui code les nombres entiers $\mathbf{Z}$ admet une telle fonction caractéristique, qu'on nomme, suivant la terminologie Lisp, `integerp`. De même, toutes les $\mathcal{U}$-classes *finies* vérifient cette propriété. Par exemple, la $\mathcal{U}$-classe $\mathcal{B} = \{0, 1, 2\}$, admet comme fonction caractéristique :

```
(lambda (terme) (member terme (0 1 2))).
```

   Par contre, une $\mathcal{U}$-classe comme $\Phi(\mathcal{Z}, \mathbf{Bool})$ n'a pas de fonction caractéristique *dans* $\mathcal{U}$ ; ceci est une simple conséquence du paradoxe de Russell.

2. De même, une relation d'équivalence sur une telle $\mathcal{U}$-classe $\mathcal{B}$ doit être définie à l'aide d'un terme du $\lambda$-calcul. Ainsi, étant donné un quotient $Q = \mathcal{B}/\sim$, il doit exister un terme $\mathtt{e} \in \Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ tel que, si $x$

et $y$ sont des objets de $\mathcal{B}$, alors $x \sim y$ si et seulement si $\Gamma_{\mathtt{e}}(x, y) = \mathit{true}$. Ici le terme `e` est à comprendre comme une "fonction" à deux arguments ; autrement dit, $\Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ est une abréviation pour $\Phi(\mathcal{B}, \Phi(\mathcal{B}, \mathbf{Bool}))$.

Dans une telle situation, on écrira $Q = \mathcal{B}/\Gamma_{\mathtt{e}}$.

Les deux informations précédemment décrites (la fonction caractéristique `b` et la relation d'équivalence `e`) suffisent pour coder les quotients de $\mathcal{U}$-classes qui nous intéressent.

Si on note les structures par `#s(`*champ-1 composante-1* ... `)`, on choisit comme code pour $\mathcal{B}/\Gamma_{\mathtt{e}}$ la structure `#s(classe b equivalence e)`, où $\mathcal{B} = \{\tau \in \mathcal{U} \ ; \ \Gamma_{\mathtt{b}}(\tau) = \mathit{true}\}$ et $\mathtt{e} \in \Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ vérifie les propriétés habituelles des relations d'équivalence.

Par exemple, l'ensemble des entiers modulo 3 peut être codé par

```
#s(classe  integerp
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3))))
```

Ce codage (non-relationnel) des quotients de $\mathcal{U}$-classes est d'une nature "fonctionnelle". Comme il a été déjà remarqué dans le paragraphe précédent, cette sorte de codage est de portée très générale, mais il a ses limites. En particulier, aucun algorithme général ne peut déterminer *sur notre machine* $\mathcal{U}$ la cardinalité de l'ensemble codé.

Mais, pour certains quotients, les renseignements connus sont beaucoup plus précis et il est judicieux d'en tenir compte dans le codage utilisé.

Par exemple, l'ensemble $\{0, 1, 2\}$ est un *ensemble complet de représentants* pour les entiers modulo 3. Dans une telle situation, il est préférable et même souvent indispensable de garder cet ensemble de représentants dans le codage.

Ainsi, on peut imaginer un nouveau codage des quotients de $\mathcal{U}$-classes dont les codes sont des structures à trois champs : `classe`, `equivalence` comme précédemment et de plus un troisième champ `representants` contenant une *liste* dont les éléments définissent un ensemble complet de représentants pour l'ensemble codé. Par exemple, un code possible pour les entiers modulo 3 est :

```
#s(classe  integerp
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3)))
   representants (0 1 2))
```

Evidemment il est possible de coder beaucoup moins de quotients de cette façon que dans le premier codage défini ; l'ensemble quotient doit en effet nécessairement être *fini* pour pouvoir être ainsi codé. Par contre, il est possible, entre autres, de calculer *dans* $\mathcal{U}$ la cardinalité de l'ensemble codé, comme il le montre le combinateur suivant, décrit en (pseudo) Lisp :

```
(lambda (quotient)
  (length (quotient-representants quotient)))
```

où `length` est le terme qui sert à calculer la longueur d'une liste.

L'expérience montre qu'en fait le champ `classe` n'est jamais utilisé lorsqu'il s'agit de programmer des algorithmes manipulant les quotients de $\mathcal{U}$-classes. Il a donc finalement été décidé de coder les quotients de $\mathcal{U}$-classes à l'aide de structures à deux champs : `representants` (une liste) et `equivalence` (la définition de la relation d'équivalence).

Ces codes permettent de travailler sur machine avec les quotients de $\mathcal{U}$-classes exactement comme les codes utilisés plus haut (compte-tenu du fait qu'on ne se sert jamais *sur machine* de la composante `classe`). Le problème maintenant est qu'*on ne peut déterminer d'une façon naturelle la structure complète "quotient" (notamment l'ensemble d'origine) d'une* $\mathcal{U}$*-classe à partir d'un tel code.* Soit, par exemple, $\mathcal{B}_1 = \mathcal{Z}$ (la $\mathcal{U}$-classe des entiers) et $\mathcal{B}_2 = \{0, 1, 2, 3, 4\}$. Il est évident que la structure

```
#s(representants (0 1 2)
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3))))
```

code aussi bien le quotient $\mathcal{B}_1/\sim$ que le quotient $\mathcal{B}_2/\sim$, où $\sim$ est la relation "être égal modulo 3". Comment maintenir la souplesse de la nouvelle technique de codage, sans perdre le lien logique entre le code et l'ensemble codé ?

La solution consiste à définir un *codage relationnel* de l'ensemble $C$ des quotients de $\mathcal{U}$-classes. Soit $\mathcal{C}$ la $\mathcal{U}$-classe constituée par les structures à deux champs : `representants` et `equivalence`. On définit alors un codage relationnel $\mathcal{R} \subset \mathcal{C} \times C$ comme suit. Une paire $(\texttt{\#s(representants l equivalence e)}, Q) \in \mathcal{R}$ si et seulement si :

- il existe une $\mathcal{U}$-classe $\mathcal{B}$ telle que $\texttt{e} \in \Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ et $Q = \mathcal{B}/\Gamma_{\texttt{e}}$.
- si $\texttt{l} = (x_0 \ldots x_i \ldots x_n)$, l'ensemble $\{x_0, \ldots, x_i, \ldots, x_n\}$ est contenu dans $\mathcal{B}$ et $\Gamma_{\texttt{e}}(x_i, x_j) = \mathit{true}$ si et seulement si $i = j$ (cette propriété implique que la liste `l` contient des éléments non-congruents deux à deux par rapport à la relation d'équivalence).
- la longueur de `l` est égale à la cardinalité de $Q$ (autrement dit, les éléments de `l` constituent un système *complet* de représentants).

Les deux propriétés essentielles des codages relationnels sont :

1. Un seul terme peut coder des objets différents. Un exemple de ce type a déjà été donné pour illustrer ce type de codage. Remarquer que, dans le cas des quotients de $\mathcal{U}$-classes, si un même terme $\tau$ code deux quotients $Q_1$ , $Q_2$ (autrement dit, $(\tau, Q_1) \in \mathcal{R}$ et $(\tau, Q_2) \in \mathcal{R}$), on déduit une bijection canonique entre $Q_1$ et $Q_2$.
2. Un objet peut être codé par des termes différents (le codage n'est pas univoque, suivant la terminologie du paragraphe 2). Par exemple,

```
#s(representants (0 1 2)
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3))))
```

et

```
#s(representants (37 14 21)
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3))))
```

sont deux codes pour les entiers modulo 3.

Un dernier exemple : la $\mathcal{U}$-classe $\{0, 1, 2\}$ (quotient trivial) est codée aussi bien par la structure :

```
#s(representants (0 1 2)
   equivalence  (lambda (n m)
                  (= (mod n 3) (mod m 3))))
```

que par la structure :

```
#s(representants (0 1 2)
   equivalence  =)
```

Considérons maintenant le quotient de la $\mathcal{U}$-classe constituée par *toutes* les listes où deux listes sont équivalentes si elles ont la même longueur. Ce quotient ne peut pas être codé comme précédemment, car sa cardinalité est infinie. D'où la nécessité d'introduire un type de codage plus général (toujours relationnel) de l'ensemble des quotients de $\mathcal{U}$-classes. D'abord quelques notations.

**Définition 1.3.1** *Un terme $\tau \in \mathcal{U}$ est nommé une* équivalence par rapport à *une $\mathcal{U}$-classe $\mathcal{B}$ si $\tau \in \Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ et si $\Gamma_\tau$ définit une relation d'équivalence sur $\mathcal{B}$.*

Remarquer que si $\tau$ est une équivalence par rapport à $\mathcal{B}$, alors $\tau$ est aussi une équivalence pour n'importe quelle sous-classe $\mathcal{B}_1$ de $\mathcal{B}$.

Définissons alors un dernier codage pour l'ensemble $C$ des quotients de $\mathcal{U}$-classes. La $\mathcal{U}$-classe $\mathcal{C}$ est constituée des termes qui sont des équivalences par rapport à une $\mathcal{U}$-classe (non précisée) au sens introduit plus haut. Le codage relationnel $\mathcal{R} \subset \mathcal{C} \times C$ est alors défini comme suit : $(\tau, Q) \in \mathcal{R}$ si et seulement si $\tau$ est une équivalence par rapport à une $\mathcal{U}$-classe $\mathcal{B}$ et $Q = \mathcal{B}/\Gamma_\tau$. Autrement dit, étant donné une paire $(\tau, Q)$ élément de $\mathcal{C} \times C$, pour assurer que $(\tau, Q) \in \mathcal{R}$ il faut déterminer une $\mathcal{U}$-classe $\mathcal{B}$ ayant les trois propriétés suivantes :

- $\tau$ est un élément de $\Phi(\mathcal{B} \times \mathcal{B}, \mathbf{Bool})$ ;
- $\Gamma_\tau$ définit une relation d'équivalence sur $\mathcal{B}$ ;
- enfin, $Q$ est le quotient $\mathcal{B}/\Gamma_\tau$.

Ainsi les quotients de $\mathcal{U}$-classes considérés dans les exemples précédents sont codés dans ce nouveau cadre *seulement par* une *équivalence*. Exemples :

1. pour les listes modulo la relation de même longueur, un code possible $\tau$ est

```
(lambda (liste1 liste2)
     (= (length liste1) (length liste2)))
```

   et $\mathcal{B}$, non "visible" dans le codage, est la $\mathcal{U}$-classe de toutes les listes.

2. pour les entiers modulo trois, l'équivalence $\tau$ est

```
(lambda (n m)
     (= (mod n 3) (mod m 3)))
```

   et la $\mathcal{U}$-classe $\mathcal{B}$ sur laquelle l'équivalence est définie est la $\mathcal{U}$-classe $\mathcal{Z}$ des entiers.

Compte tenu de la remarque faite plus haut, le dernier codage introduit a la propriété suivante : si un terme $\tau$ est un code pour un quotient $Q = \mathcal{B}/\Gamma_\tau$ il est aussi un code pour n'importe quel *sous-quotient* $Q_1 = \mathcal{B}_1/\Gamma_\tau$, où $\mathcal{B}_1 \subset \mathcal{B}$.

Par exemple, un code pour la $\mathcal{U}$-classe des entiers $\mathcal{Z}$ (quotient trivial) est le combinateur `=`. Ce même combinateur code aussi les nombres pairs, impairs, ... Cette propriété peut paraître un peu paradoxale, mais la technique du codage relationnel permet de gérer ces situations sans problème majeur.

Résumons la discussion entreprise dans cette sous-section. On a introduit deux codages relationnels différents pour l'ensemble des quotients de $\mathcal{U}$-classes :

1. Dans le premier codage, les codes sont des structures à deux champs : `representants` (contenant une liste complète de représentants du quotient) et `equivalence` (qui est la définition sur machine de la relation définissant le quotient).

2. Le deuxième codage est beaucoup plus général : il consiste à ne garder sur machine que la relation d'équivalence.

Maintenant, on peut commencer la discussion sur le codage des groupes abéliens libres.

### 1.3.2 Groupes abéliens libres.

Comme il a été déjà indiqué au début de cette section, les groupes abéliens libres à traiter sont toujours à base distinguée et, même plus, à base un quotient d'une $\mathcal{U}$-classe. Toutes les remarques faites sur le codage de ces quotients s'appliquent donc ici. Mais, d'abord, on va montrer que pour travailler sur machine avec ces groupes il suffit de connaître un code de la base.

Rappelons que dans les deux codages relationnels possibles des quotients de $\mathcal{U}$-classes, il y a toujours un élément présent : un terme $\tau \in \Phi(\mathcal{B}\times\mathcal{B}, \mathbf{Bool})$ définissant une relation d'équivalence. Par exemple, l'ensemble des entiers modulo 3 contient dans son code le terme :

```
(lambda (n m)
  (= (mod n 3) (mod m 3))))
```

Etant donné $\tau$ et $Q = \mathcal{B}/\Gamma_\tau$, on définit alors la $\mathcal{U}$-classe des éléments du groupe engendré par $Q$ comme l'ensemble des listes dont chaque composante est à son tour constitué d'un élément de $\mathcal{Z}$ (le coefficient) et d'un autre élément de $\mathcal{B}$ (le générateur). Par exemple, des éléments du groupe abélien libre engrendré par les entiers modulo 3 (noté habituellement $\mathbf{Z}[\mathbf{Z_3}]$) sont :

```
((-2 1) (127 0) (-1 5))
((1 0) (-1 1))
((7 2))
```

Si, pour éviter des ambiguïtés, on note $x_0$, $x_1$, $x_2$ les éléments de $\mathbf{Z_3}$ et on emploie les notations usuelles pour les éléments de $\mathbf{Z}[\mathbf{Z_3}]$, ces listes codent repectivement : $-2x_1 + 127x_0 - x_2$, $x_0 - x_1$ et $7x_2$.

Cette représentation étant fixée une fois pour toutes, la fonction de changement de signe et l'élément neutre (la liste vide) sont indépendants du groupe abélien libre particulier en cours de traitement. Par exemple, l'opposé de

```
((-2 1) (127 0) (-1 5))
```

est

```
((2 1) (-127 0) (1 5))
```

Par ailleurs, la fonction somme ne dépend que du test d'égalité entre générateurs, autrement dit, du terme qui code la relation d'équivalence. Dans l'exemple du codage de $\mathbf{Z}[\mathbf{Z_3}]$, la somme de `((-2 1) (127 0) (-1 5))` et de `((7 2))` donne l'élément `((-2 1) (127 0) (6 5))`, où les seuls renseignements utilisés sont : $\Gamma_\tau(1,2) = \Gamma_\tau(0,2) = \mathit{false}$ et $\Gamma_\tau(5,2) = \mathit{true}$ (le terme $\tau$ étant l'équivalence définie plus haut).

On pourrait aussi utiliser une fonction caractéristique de la $\mathcal{U}$-classe $\mathcal{B}$ pour vérifier que les listes précédentes sont de *vrais* éléments du groupe codé, mais l'expérience montre qu'en programmation concrète, une telle information n'est jamais utilisée. D'où le choix du codage relationnel pour les bases de groupes, codage valable aussi bien pour le groupe déduit de cette base. Comme pour les quotients de $\mathcal{U}$-classes, deux codages relationnels sont utilisés :

1. Le codage des groupes abéliens libres de type fini ; autrement dit, des groupes dont la base est un ensemble fini.

   On décide que la $\mathcal{U}$-classe $\mathcal{C}$ choisie pour définir le codage est constituée de structures à deux champs : `base`, liste complète de représentants des éléments de la base (rappelons que les bases sont des quotients de $\mathcal{U}$-classes) et `egl`, définition d'une relation d'équivalence (à interpréter dans ce contexte comme le code du test d'égalité entre les générateurs du groupe). Par exemple, un code possible pour $\mathbf{Z}[\mathbf{Z_3}]$ serait :

```
#s(base (0 1 2)
   egl  (lambda (n m)
          (= (mod n 3) (mod m 3))))
```

2. Les codages des groupes abéliens (non supposés de type fini). Dans ce cas on se contente de garder en machine le test d'égalité entre les générateurs du groupe. On explicite ce codage relationnel.

   Etant donné la classe $C$ des groupes abéliens libres et la $\mathcal{U}$-classe $\mathcal{C}$ des *équivalences* (dans le sens défini dans la sous-section précédente), on définit un sous-ensemble $\mathcal{R}$ de $\mathcal{C} \times C$ comme suit. Une paire $(\tau, G) \in \mathcal{R}$ si et seulement si :

- Il existe une $\mathcal{U}$-classe $\mathcal{B}$ et une relation d'équivalence sur $\mathcal{B}$, tels que l'ensemble quotient $B$ est la base (distinguée) de $G$ ; autrement dit, $G = \mathbf{Z}[B]$.
- De plus, $\tau$ est une équivalence par rapport à la $\mathcal{U}$-classe $\mathcal{B}$ et $B = \mathcal{B}/\Gamma_\tau$.

Par exemple, le groupe abélien libre engendré par les listes modulo la relation "être de même longueur" est codé par le terme :

```
(lambda (liste1 liste2)
   (= (length liste1) (length liste2)))
```

De même, le groupe abélien libre $\mathbf{Z}[\mathbf{Z}]$ est codé tout simplement par le combinateur `=`. Remarquer aussi que le groupe $\mathbf{Z}[2\mathbf{Z}]$, où $2\mathbf{Z}$ est l'ensemble des nombres entiers pairs, admet comme code le même test `=`.

La différence essentielle entre le codage des groupes de type fini et celui des groupes en général, c'est que dans le dernier cas on ne peut obtenir sur notre machine théorique que des renseignements de nature "locale". On peut par exemple répondre à une question comme : "étant donné deux générateurs d'un groupe, sont-ils égaux ?". Par contre, on ne pourra pas en général répondre (dans le cadre de notre machine $\mathcal{U}$) à des questions "globales" comme : étant donné le code d'un groupe, est-il de type fini ?".

Par contre, si on travaille avec le premier codage relationnel donné, le champ `base` peut être utilisé pour définir une fonction dans $\mathcal{U}$ (un terme du $\lambda$-calcul) calculant la dimension du groupe :

```
(lambda (code)
   (length code-base))
```

Cette dimension détermine la classe d'isomorphisme du groupe (si la dimension est $n$, le groupe codé est isomorphe à $\mathbf{Z}^n$), renseignement nécessaire pour les calculs d'algèbre homologique. Bien entendu, le calcul de la dimension est un problème de nature "globale" et il ne peut pas exister un terme calculant la dimension à partir du code dans le deuxième codage introduit.

Compte tenu de ces remarques, on dira que le premier codage pour les groupes de type fini (où les bases sont codées à l'aide de listes) est un *codage effectif*, tandis que le deuxième codage est un *codage localement effectif*.

En pratique, pour garder le maximun de cohérence et de simplicité, on décide que le code d'un groupe est toujours une structure `#s(base comp1 egl comp2)`, où `comp1` peut être une liste (dans le cas où une base finie est connue ; ce cas correspond aux groupes qu'on pourrait coder dans le codage 1 défini plus haut) ou bien le symbole `:locally-effective` (dans le cas où on ne dispose d'aucun renseignement particulier sur la cardinalité ou les représentants de la base ; ces groupes seraient codés par la technique 2). Par exemple, un code pour $\mathbf{Z}[\mathbf{Z}_3]$ serait :

```
#s(base (0 1 2)
   egl  (lambda (n m)
          (= (mod n 3) (mod m 3))))
```

tandis que le code pour $\mathbf{Z}[\mathbf{Z}]$ (ou $\mathbf{Z}[2\mathbf{Z}], \ldots$) serait

```
#s(base :locally-effective
   egl  =)
```

On va démontrer maintenant à titre d'exemple que l'opération de somme directe de deux groupes abéliens libres est calculable (au sens défini dans le paragraphe 2) par rapport à ce codage relationnel.

Soient `#s(base b1 egl e1)`, `#s(base b2 egl e2)` les codes des deux groupes $G_1 = \mathbf{Z}[B_1]$ et $G_2 = \mathbf{Z}[B_2]$. On veut construire un nouveau code `#s(base b egl e)` tel que (`#s(base b egl e)`,$G$) $\in \mathcal{R}$, où $G$ est un groupe isomorphe à $G_1 \oplus G_2$.

Pour ce faire, on considère une nouvelle structure à deux champs : `ind` qui contiendra un indice, le nombre 1 ou 2, et `gen` qui contiendra un élément de $\mathcal{U}$.

La base `b` sera définie comme le symbole `:locally-effective` lorsque `b1` ou `b2` (ou les deux) est ce symbole ; autrement dit, le codage de $G_1 \oplus G_2$ est effectif si et seulement si les codages de $G_1$ et $G_2$ sont effectifs. Dans ce dernier cas, on construit la nouvelle base à partir des listes des générateurs de $G_1$ et $G_2$. Le test d'égalité `e` est défini de façon évidente.

Le point important dans cette construction est que le passage entre les codes `#s(base b1 egl e1)`, `#s(base b2 egl e2)` et le code `#s(base`

b egl e) peut être complètement décrit dans notre machine théorique. Autrement dit, il existe un terme du $\lambda$-calcul (un combinateur) qui code la construction mathématique "somme directe".

Par exemple cette fonction (pseudo) Lisp accomplit cette tâche :

```
(lambda (code1 code2)
  (make-code
   :base (if (or (equal code1-base :locally-effective)
                 (equal code2-base :locally-effective))
             :locally-effective
             (construire-liste-gen code1-base code2-base))
   :egl  (lambda (g1 g2)
           (and (= g1-ind g2-ind)
                (if (= 1 g1-ind)
                    (code1-egl g1-gen g2-gen)
                    (code2-egl g1-gen g2-gen))))))
```

Les objets essentiels de l'algèbre homologique sont, bien entendu, les complexes de chaînes. Dans notre contexte un complexe de chaînes est un groupe abélien *libre* $C$ à base graduée muni d'un opérateur de bord $d : C \to C$ de degré $-1$ vérifiant $d \circ d = 0$. Compte tenu de la discussion précédente sur le codage des groupes abéliens libres, on se contente de caractériser la $\mathcal{U}$-classe associée au codage relationnel des complexes de chaînes. On laisse le lecteur déterminer un codage pour les autres objets usuels en topologie algébrique.

Le code pour un complexe de chaînes va être une structure avec les champs suivants :

- `egl`, un test d'égalité entre les générateurs.
- `cbs`, ou bien le symbole `:locally-effective` (lorsqu'on ne dispose d'aucun renseignement sur la cardinalité ou la nature des générateurs du groupe), ou bien un terme appartenant à $\Phi(\mathcal{Z}, \mathcal{L})$, où $\mathcal{L}$ est la $\mathcal{U}$-classe de toutes les listes (on doit alors interpréter cette "fonction" comme l'algorithme qui, étant donné un entier $n$, calcule la liste des générateurs en degré $n$ du groupe en question).
- `d`, la fonction de bord.

Selon le contenu de la composante `cbs` on dira que (le codage de) un complexe de chaînes est *effectif* ou *localement effectif*. L'une des propriétés essentielles des complexes de chaînes effectifs est l'existence d'un algorithme (un terme du $\lambda$-calcul) calculant leurs groupes d'homologie (en chaque dimension). Bien entendu, il n'existe pas un tel algorithme pour les complexes localement effectifs, même si on sait par des raisons théoriques que le complexe codé est à homologie de type fini. Mais les méthodes de l'homologie effective (Sergeraert [31]) permettent néanmoins de tourner souvent cette difficulté.

# 2 Homologie effective.

## 2.1 Définitions, propriétés et notations.

Dans tout ce travail, les complexes de chaînes sont *libres* et *nuls en les degrés négatifs*. Plus précisement, un *complexe de chaînes* est un couple $(C, d)$ où $C = \{C_n\}_{n \in \mathbf{Z}}$ est un module gradué, chaque $C_n$ est un $\mathbf{Z}$-module libre à base distinguée, $d$ est une différentielle, $C_n = 0$ si $n < 0$ et $d_n = 0$ si $n \leq 0$.

Si on veut être rigoureux, on doit travailler avec les codages des complexes de chaînes tels qu'ils ont été introduits dans la section 1.3. Mais, pour ne pas trop alourdir la rédaction, on se contentara de la terminologie usuelle, en supposant que les exemples presentés dans le premier chapitre suffisent pour que le lecteur traduise nos énoncés dans le cadre d'une machine théorique.

En particulier, il est important de noter les points suivants :

1. Par "complexe de chaînes" il faut entendre "complexe de chaînes admettant un codage (effectif ou localement effectif) sur notre machine théorique" ; et surtout :

2. Un complexe de *type fini* est un complexe à codage *effectif*.

**Définition 2.1.1** *Une* réduction *est un ensemble de données* $\{C, C', f, g, h\}$ *où* $f : C \to C'$, $g : C' \to C$ *sont des morphismes de complexes de chaînes et* $h : C \to C$ *est un opérateur d'homotopie ; les conditions suivantes sont demandées :*

**(r1)** $fg = 1_{C'}$;
**(r2)** $fh = 0$;
**(r3)** $hg = 0$;
**(r4)** $hd + dh + gf = 1_C$;
**(r5)** $hh = 0$.

Dans cette définition on a suivi la terminologie de Sergeraert [31] ; on trouve aussi dans la littérature : "strong deformation retraction" ou $SDR$ chez Lambe [22] ou simplement "retraction" chez Eilenberg-MacLane [13].

**Proposition 2.1.2** *Le composé de deux réductions est une autre réduction.*

Dans ce mémoire, chaque énoncé sera suivi par une spécification des entrées et sorties de l'algorithme correspondant :

**Algorithme 2.1.3**
Entrées : *deux réductions* $\{C_2, C_1, f_1, g_1, h_1\}$, $\{C_1, C_0, f_2, g_2, h_2\}$ ;
Sortie : *une réduction* $\{C_2, C_0, f, g, h\}$.

Les algorithmes reliant les entrées et sorties seront décrits dans le texte de l'une des trois façons suivantes:

1. Dans l'énoncé même. Par example, si on énoncé la proposition précédente de cette manière :

   **Proposition.** Étant donné deux réductions $\{C_2, C_1, f_1, g_1, h_1\}$, $\{C_1, C_0, f_2, g_2, h_2\}$, on peut en déduire une troisième $\{C_2, C_0, f, g, h\}$, où

   $$f := f_2 f_1;$$
   $$g := g_1 g_2;$$
   $$h := h_1 + g_1 h_2 f_1.$$

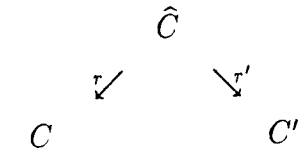
2. La propriété est évidente et décrire l'algorithme est laissé au lecteur. C'est le cas pour ce résultat choisi à titre exemple.

3. La description de l'algorithme est esquissée dans la démonstration donnée du résultat.

La véritable traduction de l'algorithme sur une machine théorique ou concrète est laissée en exercice, puisqu'elle est directe. Par exemple, on donne la fonction Lisp implémentant la proposition 2.1.2.

```
(DEFUN CMP-RDC (brdc trdc) ; bottom-rdc top-rdc
  (let ((tf (rdc-f trdc))
        (tg (rdc-g trdc)))
    (build-rdc
      :f (cmp-mrp (rdc-f brdc) tf)
      :g (cmp-mrp tg (rdc-g brdc))
      :h (add-mrp-to-mrp (rdc-h trdc)
                         (cmp-mrp tg
                                  (cmp-mrp (rdc-h brdc)
                                           tf))))))
```

**Définition 2.1.4** *Une* équivalence d'homotopie *entre deux complexes de chaînes $C$ et $C'$ est une paire de réductions $\{r, r'\}$ d'un même complexe $\hat{C}$ vers $C$ et $C'$.*




Plus précisement, une équivalence d'homotopie (entre $C$ et $C'$) est un ensemble de données $\{C, C', \hat{C}, f_C, g_C, h_C, f_{C'}, g_{C'}, h_{C'}\}$ où $\{\hat{C}, C, f_C, g_C, h_C\}$, $\{\hat{C}, C', f_{C'}, g_{C'}, h_{C'}\}$ sont des réductions.

**Remarque 2.1.5** *Cette définition est équivalente à la définition habituelle si dans celle-ci on demande de plus que l'inverse homotopique et les homotopies soient données explicitement : il suffit alors de prendre pour $\hat{C}$ le "mapping cylinder" de l'équivalence d'homotopie (voir une démonstration de ce fait dans [4]).*

**Définition 2.1.6** *L'*homologie effective *d'un complexe de chaînes $C$ est un ensemble de données $\{C, HC, \omega\}$ où $\omega$ est une équivalence d'homotopie entre $C$ et un complexe de chaînes de type fini $HC$. Ce triplet $\{C, HC, \omega\}$ est nommé* un complexe de chaînes à homologie effective.

Il y a bien unicité à une équivalence près facile à définir.

Une première propriété des complexes à homologie effective est que son homologie ordinaire est calculable : cette homologie n'est autre que celle du complexe de type fini $HC$.

**Algorithme 2.1.7**

Entrées :

- *un complexe à homologie effective $\{C, HC, \omega\}$ ;*
- *un nombre entier positif $n$ ;*

Sortie : *le $n$-ième groupe d'homologie de $C$.*

Une deuxième conséquence immédiate est que tout complexe de type fini est (trivialement) à homologie effective.

**Algorithme 2.1.8**

Entrée : *un complexe de chaînes $C$ de type fini ;*

Sortie : *un complexe de chaînes à homologie effective $\{C, C, Id\}$.*

**Proposition 2.1.9** *Soient $\{\widehat{X}^1, X^1, f^1, g^1, h^1\}$, ... , $\{\widehat{X}^n, X^n, f^n, g^n, h^n\}$, $n$ réductions. Alors, on peut en déduire une autre réduction $\{\widehat{X}^1 \otimes \ldots \otimes \widehat{X}^n$, $X^1 \otimes \ldots \otimes X^n, f, g, h\}$ où*

$$f := f^1 \otimes \ldots \otimes f^n;$$
$$g := g^1 \otimes \ldots \otimes g^n;$$
$$h := \sum_{i=1}^{n} 1 \otimes \ldots \otimes 1 \otimes h^i \otimes g^{i+1} f^{i+1} \otimes \ldots \otimes g^n f^n.$$

**Algorithme 2.1.10**

Entrées : *$n$ réductions $\{\widehat{X}^1, X^1, f^1, g^1, h^1\}$, ... , $\{\widehat{X}^n, X^n, f^n, g^n, h^n\}$ ;*

Sortie : *une réduction $\{\widehat{X}^1 \otimes \ldots \otimes \widehat{X}^n, X^1 \otimes \ldots \otimes X^n, f, g, h\}$.*

**Remarque 2.1.11 Convention sur les signes dans les produits tensoriels**

*Soit $f^i$ un morphisme de degré $n_i$ et $a^j$ un générateur de degré $m_j$. Alors chaque fois que $f^i$ "saute" un élément $a^j$ il faut multiplier le signe global par $(-1)^{n_i m_j}$. Par exemple :*

$$(f^1 \otimes f^2)(a^1 \otimes a^2) := (-1)^{n_2 m_1} f^1(a^1) \otimes f^2(a^2);$$
$$(f^1 \otimes f^2 \otimes f^3)(a^1 \otimes a^2 \otimes a^3) := (-1)^{n_3 m_1 + n_3 m_2 + n_2 m_1} f^1(a^1) \otimes f^2(a^2) \otimes f^3(a^3).$$

*Ces conventions s'appliquent pour les différentielles, les morphismes de chaînes, les opérateurs d'homotopie et ainsi de suite. Il faut en tenir compte dans l'énoncé précédent.*

**Corollaire 2.1.12** *Si $X^1, \ldots, X^n$ sont des complexes de chaînes à homologie effective, alors $X^1 \otimes \ldots \otimes X^n$ est un complexe de chaînes à homologie effective.*

**Algorithme 2.1.13**

Entrées : *$n$ complexes à homologie effective $\{X^1, HX^1, \omega^1\}$, ... , $\{X^n, HX^n, \omega^n\}$ ;*

Sortie : *un complexe à homologie effective $\{X^1 \otimes \ldots \otimes X^n, HX^1 \otimes \ldots \otimes HX^n, \omega\}$.*

## 2.2 La théorie de perturbation homologique.

Soit $(C, d)$ un complexe de chaînes et supposons qu'on y modifie la différentielle pour obtenir un nouveau complexe de chaînes $(C, d')$ ; quelle relation peut-on établir entre les homologies de ces deux complexes ? Par exemple, si $H_*(C, d)$ est de type fini, sous quelles conditions $H_*(C, d')$ est aussi de type fini ? Ces questions sont agréablement traitées par les méthodes dites de perturbation homologique (le nouveau complexe peut en effet être interprété comme le complexe de départ dont la différentielle a été "perturbée" par l'opérateur $d' - d$).

L'outil le plus important dans cet ordre d'idées est connu sous le nom de "Basic Perturbation Lemma" ; il fournit un cadre très large où la question posée plus haut a, moyennant de bonnes hypothèses, une réponse affirmative.

Ce "lemme de perturbation" a été initialement introduit par Shih [33] pour démontrer le théorème d'Eilenberg-Zilber tordu (ce cas est traité en détail dans le chapitre suivant). Il a été plus tard systematisé par R. Brown [10], puis par Gugenheim [17]. Il a connu récemment de nombreux développements (voir, par exemple, [22], [18], [23], [19]).

**Définition 2.2.1** *Soient $C$ un module gradué et $f : C \to C$ un morphisme de modules gradués. Le morphisme $f$ est* localement nilpotent *si pour tout $x$, élément non nul de $C$, il existe un nombre entier positif $n$ (en général, le nombre $n$ dépend de l'élément $x$) tel que $f^n(x) = 0$, où $f^n : C \to C$ est le morphisme de modules gradués obtenu en itérant $n$ fois le morphisme $f : C \to C$.*

Dans le cadre des applications du lemme de perturbation, pour démontrer qu'un morphisme est localement nilpotent, on utilise fréquemment que les modules gradués envisagés sont munis des filtrations. Rappelons qu'une *filtration* d'un module gradué $C$ est une suite de modules :

$$\{0\} = C^{(-1)} \subset C^{(0)} \subset \ldots \subset C^{(n)} \subset \ldots \subset C,$$

où

$$C = \bigcup_{n=0}^{\infty} C^{(n)}.$$

Il est évident que si un morphisme $f : C \to C$ diminue l'indice de filtration dans $C$, alors $f$ est un morphisme localement nilpotent.

**Définition 2.2.2** *Une* perturbation *d'un complexe de chaînes $(C,d)$ est un morphisme de modules gradués $\rho : C \to C$, de degré $-1$, et tel que $(d+\rho)^2 = 0$.*

**Théorème 2.2.3 Lemme de Perturbation**

*Soit $\{(A,d_A),(B,d_B),f,g,h\}$ une réduction et soit $\rho : A \to A$ une perturbation de $(A,d_A)$ telle que le composé $h\rho : A \to A$* est un morphisme localement nilpotent. *Alors une nouvelle réduction $\{(A,d_A+\rho),(B,d_\infty),f_\infty,g_\infty,h_\infty\}$ est définie par les formules :*

$$\begin{aligned} d_\infty &:= d_B + f\rho\Sigma_\infty g \ ; \\ f_\infty &:= f(1-\rho\Sigma_\infty h) \ ; \\ g_\infty &:= \Sigma_\infty g \ ; \\ h_\infty &:= \Sigma_\infty h, \end{aligned}$$

*où* $\Sigma_\infty := 1 - h\rho + h\rho h\rho - \ldots + (-1)^i(h\rho)^i + \ldots$.

**Algorithme 2.2.4**

Entrées :

- *une réduction $\{(A,d_A),(B,d_B),f,g,h\}$ ;*
- *une perturbation $\rho$ de $(A,d_A)$ ;*

Sortie : *une réduction $\{(A,d_A+\rho),(B,d_\infty),f_\infty,g_\infty,h_\infty\}$.*

Dans cet énoncé, l'application $\Sigma_\infty : A \to A$ qui est décrite comme une série infinie est en fait une somme finie lorsqu'on l'applique sur un élément donné, car $h\rho : A \to A$ est un morphisme localement nilpotent. Compte tenu de cette propriété, il est facile de vérifier que les données $\{(A,d_A+\rho),(B,d_\infty),f_\infty,g_\infty,h_\infty\}$ définissent une réduction.

Est capital dans ce résultat le fait que les perturbations interviennent au niveau des morphismes et des différentielles alors que les modules gradués restent inchangés. En particulier, si le complexe $B$ est de type fini, on a là une méthode de portée très générale pour construire des complexes à homologie effective. Cette méthode est détaillée dans le théorème suivant. Introduisons d'abord une notion qui va faciliter son énoncé.

**Définition 2.2.5** *Soit* $\{C, HC, \widehat{C}, f_C, g_C, h_C, f_{HC}, g_{HC}, h_{HC}\}$ *l'homologie effective d'un complexe de chaînes* $C$ *et soit* $\rho : C \to C$ *une perturbation de* $C$. *Cette homologie effective et la perturbation* $\rho$ *sont* compatibles *si l'application composée* $h_{HC} g_C \rho f_C : \widehat{C} \to \widehat{C}$ *est un morphisme localement nilpotent.*

**Théorème 2.2.6** *Soient* $(C, d)$ *un complexe de chaînes à homologie effective et une perturbation* $\rho : C \to C$ *de la différentielle qui soient compatibles. Alors, le complexe de chaînes* $(C, d + \rho)$ *est aussi à homologie effective.*

**Démonstration.**

Soit $\{C, HC, \widehat{C}, f_C, g_C, h_C, f_{HC}, g_{HC}, h_{HC}\}$ l'homologie effective donnée de $C$. À partir de la réduction $\{(\widehat{C}, \widehat{d}), (C, d), f_C, g_C, h_C\}$, une nouvelle réduction $\{(\widehat{C}, \widehat{d} + \widehat{\rho}), (C, d + \rho), f_C, g_C, h_C\}$ vers $(C, d + \rho)$ est définie, où $\widehat{\rho} := g_C \rho f_C : \widehat{C} \to \widehat{C}$ est une perturbation de $(\widehat{C}, \widehat{d})$.

Maintenant, l'homologie effective de départ et $\rho$ sont compatibles, donc le composé $h_{HC}\widehat{\rho} : \widehat{C} \to \widehat{C}$ est un morphisme localement nilpotent. En appliquant le lemme de perturbation à $\{\widehat{C}, HC, f_{HC}, g_{HC}, h_{HC}\}$ et $\widehat{\rho}$, on en obtient une réduction de $(\widehat{C}, \widehat{d} + \widehat{\rho})$ vers un complexe de chaînes de type fini.

**Algorithme 2.2.7**

Entrées :

- *un complexe à homologie effective* $\{(C, d), HC, \omega\}$ ;
- *une perturbation* $\rho$ *de* $(C, d)$ *compatible ;*

Sortie : *un complexe à homologie effective* $\{(C, d + \rho), HC^1, \omega^1\}$.

## 2.3 Homologie des bicomplexes.

On considère une structure particulière de bicomplexe, qui sera essentielle dans le cadre de la construction Cobar (voir chapitre **4**).

Rappelons qu'un *bicomplexe* est un module bigradué $\{C_{p,q}\}_{p,q\in\mathbf{Z}}$ muni des morphismes $d'_{p,q} : C_{p,q} \to C_{p-1,q}$ et $d''_{p,q} : C_{p,q} \to C_{p,q-1}$, $p, q \in \mathbf{Z}$, vérifiant : $d'd' = 0$, $d''d'' = 0$ et $d'd'' + d''d' = 0$. On définit alors le complexe de chaînes $(T(C), d)$ *totalisation* du bicomplexe, où

$$T_n(C) = \bigoplus_{p+q=n} C_{p,q}$$

et $d = d' + d''$.

**Définition 2.3.1** *Un* bicomplexe simple *("tapered" chez Eilenberg et Moore [14]) est un bicomplexe $X^0 \to X^1 \to \cdots \to X^n \to \cdots$ (où les $X^i$ sont des complexes de chaînes) tel que $(X^n)_r = 0$ lorsque $r < 2n$. L'indice $n$ de $X^n$ est compté négativement dans la bigraduation.*

Il résulte de cette définition que le module gradué $T(X)$ obtenu par totalisation d'un bicomplexe simple est de type fini en chaque degré si chaque colonne $X^i$ est de type fini : $T_n = X^0_n \oplus \cdots \oplus X^n_{2n}$.

Partons maintenant d'une suite $X$ de complexes de chaînes $X^0, \ldots, X^n, \ldots$ vérifiant la condition $(X^n)_r = 0$ si $r < 2n$. Ces données sont suffisantes pour munir la totalisation $T(X)$ d'une structure de complexe de chaînes (en considérant qu'il s'agit d'un bicomplexe où les flèches horizontales sont nulles). Si on donne maintenant en outre une suite de morphismes de complexes de chaînes $\psi_i : X^i \to X^{i+1}$ vérifiant $\psi_{i+1}\psi_i = 0$, on peut en déduire un bicomplexe simple en changeant les signes convenablement. Désormais on ne fera pas de distinction entre les bicomplexes et les suites de morphismes de complexes de chaînes.

Le but de ce paragraphe est de calculer l'homologie d'un bicomplexe à partir de celle de ses colonnes. La méthode consiste à résoudre d'abord le cas trivial d'un bicomplexe à flèches horizontales nulles, puis à "perturber" sa totalisation à l'aide des flèches horizontales $\psi_i$.

Pour ce faire il faut que, étant donné un complexe à homologie effective $C$ tel que $C_n = 0$ pour $n$ plus petit qu'un entier fixé $n_0$, on puisse reproduire cette propriété (la nullité jusqu'au degré $n_0$) sur les complexes de chaînes

de son homologie effective. Cela nous permettra de construire à partir d'un bicomplexe simple $X^0 \to \ldots \to X^n \to \ldots$ dont les colonnes sont à homologie effective, deux nouveaux bicomplexes *simples*, dont la totalisation fournira les modules gradués d'une homologie effective du bicomplexe de départ.

**Proposition 2.3.2** *Soit $C$ un complexe de chaînes à homologie effective et soit $n_0$ tel que $C_n = 0$ lorsque $n < n_0$. Alors un algorithme permet de construire une homologie effective $\{C, \widehat{C}, HC, r, r'\}$ de $C$ vérifiant $\widehat{C}_n = 0$ et $(HC)_n = 0$ si $n < n_0$.*

**Démonstration.**
Soit $\{C, \overline{C}, \overline{HC}, \overline{r}, \overline{r}'\}$ une homologie effective quelconque de $C$. On construit d'abord une réduction de $\overline{HC}$ vers un complexe $HC$ qui soit *irréductible* (dans un sens évident). Le complexe $HC$ est caracterisé, à isomorphisme près, par l'homologie ordinaire de $C$. Donc, en particulier, $(HC)_n = 0$ si $n < n_0$. On a défini de cette manière une nouvelle homologie effective $\{C, \overline{C}, HC, \overline{r}, \overline{r}''\}$ de $C$. Il suffit alors d'enlever ce qui "dépasse" avec une toute petite précaution à la frontière.

**Algorithme 2.3.3**
Entrées :

- *Un complexe à homologie effective* $\{C, \overline{C}, \overline{HC}, \overline{r}, \overline{r}'\}$ ;
- *un nombre entier* $n_0$ *tel que* $C_n = 0$ *si* $n < n_0$ ;

Sortie : *Un complexe à homologie effective* $\{C, \widehat{C}, HC, r, r'\}$ *tel que* $(\widehat{C})_n = (HC)_n = 0$ *si* $n < n_0$.

En fait cet algorithme ne sera jamais utilisé dans les applications envisagées (chapitre **4**) : la méthode itérative employée construira automatiquement des homologies effectives vérifiant les conditions de "simplicité" (nullité jusqu'à un certain degré) exigées.

Le lemme suivant s'obtient par simple somme directe. Ce lemme définit le complexe à homologie effective sur lequel nous nous proposons d'appliquer la méthode de perturbation.

**Lemme 2.3.4** *Soit $X^0, \ldots, X^n, \ldots$ une suite de complexes de chaînes à homologie effective et vérifiant $(X^n)_r = 0$ lorsque $r < 2n$. Alors leur totalisation est aussi un complexe de chaînes à homologie effective.*

La seule précaution qu'il faut prendre pour démontrer ce lemme est de remplacer, si nécessaire, les homologies effectives de départ par d'autres ayant suffisamment de degrés triviaux, en appliquant la proposition précédente ; ensuite les formules "naturelles" fournissent bien l'équivalence d'homotopie cherchée. Remarquer que les applications ainsi construites préservent la structure "par colonnes", ce qui sera important par rapport à la filtration qu'on va tout de suite définir sur la totalisation d'un bicomplexe simple.

Soit $X^0 \stackrel{\psi_0}{\to} X^1 \stackrel{\psi_1}{\to} X^2 \to \ldots \to X^n \to \ldots$ un bicomplexe simple. On filtre sa totalisation par :

$$T^{(m)} = \bigoplus_{r-2n \leq m} (X^n)_r.$$

Noter que $T^{(-1)} = 0$, car le bicomplexe est simple.

Supposons que les colonnes de ce bicomplexe soient à homologie effective. Alors les applications définies canoniquement dans le lemme précédent préservent cette filtration, sauf l'homotopie $h : (X^n)_r \to (X^n)_{r+1}$ qui augmente au plus d'*une unité* l'indice de filtration.

Par ailleurs, les flèches horizontales $\psi_n : (X^n)_r \to (X^{n+1})_r$ diminuent de *deux* unités l'indice de filtration. Donc si on prend comme complexe à homologie effective la totalisation *sans flèches horizontales* du bicomplexe de départ, on voit que la perturbation définie par les $\psi_i$ est compatible avec l'homologie effective (car le composé de l'opérateur d'homotopie et de la perturbation induite est un morphisme localement nilpotent). Le théorème 2.2.6 s'applique donc et on en obtient :

**Théorème 2.3.5** *Si chaque colonne d'un bicomplexe simple est à homologie effective, la totalisation de ce bicomplexe est aussi à homologie effective.*

**Algorithme 2.3.6**

Entrées :

- *un bicomplexe simple $X$ ;*
- *une fonction faisant correspondre à chaque entier positif $n$, une homologie effective de la $n$-ième colonne $X^n$ ;*

Sortie : *un complexe à homologie effective $\{T(X), HT(X), \omega\}$.*

# 3 Ensembles simpliciaux et le théorème d'Eilenberg-Zilber tordu.

## 3.1 Définitions et théorème d'Eilenberg-Zilber.

Étant donné un ensemble simplicial $K$, ses opérateurs de face et de dégénérescence seront notés $\partial_i$, $s_i$, respectivement. On notera $C(K)$ son complexe de chaînes *normalisé*. Un *ensemble simplicial* $K$ est *à homologie effective* si $C(K)$ est un complexe de chaînes à homologie effective.

Rappelons que, étant donné deux entiers positifs $p$ et $q$, un $(p,q)$-*shuffle* est une permutation $\pi$ de l'ensemble $\{0,1,\ldots,p+q-1\}$ vérifiant :

$$\pi(i) < \pi(j) \text{ si } \begin{cases} 0 \leq i < j \leq p-1 \text{ ou} \\ p \leq i < j \leq p+q-1. \end{cases}$$

Si on note alors

$$\begin{cases} \alpha_i = \pi(i-1) & \text{où } 0 < i \leq p \text{ et} \\ \beta_j = \pi(j+p-1) & \text{où } 0 < j \leq q\ , \end{cases}$$

les suites $\alpha = (\alpha_1,\ldots,\alpha_p)$ ou $\beta = (\beta_1,\ldots,\beta_q)$ déterminent $\pi$. On notera le $(p,q)$-shuffle $\pi$ par $(\alpha,\beta)$ ; la signature de $(\alpha,\beta)$ est

$$sg(\alpha,\beta) = \sum_{i=1}^{p} (\alpha_i - i - 1).$$

**Définitions 3.1.1** *Soient $F$, $B$ deux ensembles simpliciaux. Les opérateurs* d'Alexander-Whitney $AW_{F,B} : C(F\times B) \to C(F)\otimes C(B)$, d'Eilenberg-MacLane $EML_{F,B} : C(F)\otimes C(B) \to C(F\times B)$ *et* de Shih $SHI_{F,B} : C_*(F\times B) \to C_{*+1}(F\times B)$ *de $F$ et $B$ sont définis par les formules suivantes :*

$$AW(x_n,y_n) = \sum_{i=0}^{n} \partial_{i+1}\ldots\partial_n x_n \otimes \partial_0\ldots\partial_{i-1} y_n,$$

$$EML(x_p\otimes y_q) = \sum_{(\alpha,\beta)\in\{(p,q)-shuffles\}} (-1)^{sg(\alpha,\beta)}(s_{\beta_q}\ldots s_{\beta_1}x_p, s_{\alpha_p}\ldots s_{\alpha_1}y_q) \text{ et}$$

$$SHI(x_n,y_n) =$$

$$= \sum (-1)^{n-p-q+sg(\alpha,\beta)} \quad (s_{\beta_q+n-p-q}\ldots s_{\beta_1+n-p-q}s_{n-p-q-1}\partial_{n-q+1}\ldots\partial_n x_n,$$

$$s_{\alpha_{p+1}+n-p-q}\ldots s_{\alpha_1+n-p-q}\partial_{n-p-q}\ldots\partial_{n-q-1}y_n),$$

*où la dernière somme est définie pour tous les indices* $0 \leq q \leq n-1$, $0 \leq p \leq n-q-1$ *et* $(\alpha, \beta) \in \{(p+1,q) - shuffles\}$.

Il est bien connu que les opérateurs d'Alexander-Whitney et d'Eilenberg-MacLane sont des équivalences d'homotopie, inverses l'une de l'autre. L'opérateur de Shih introduit n'est autre que l'expression itérative de celui défini récursivement par Shih dans [33] ; cet opérateur fournit une homotopie entre le composé $EML \circ AW$ et l'identité.

Si on travaille sur les complexes de chaînes normalisés, le théorème d'Eilenberg-Zilber [15] s'énonce d'une manière plus précise :

**Théorème 3.1.2 Théorème d'Eilenberg-Zilber**

*Soient* $F$ *et* $B$ *deux ensembles simpliciaux. Alors les données* $\{C(F \times B), C(F) \otimes C(B), AW_{F,B}, EML_{F,B}, SHI_{F,B}\}$ *définissent une* réduction.

**Algorithme 3.1.3**

Entrées : *deux ensembles simpliciaux* $F$, $B$ ;

Sortie : *une réduction* $\{C(F \times B), C(F) \otimes C(B), AW_{F,B}, EML_{F,B}, SHI_{F,B}\}$.

Remarquer que ce théorème et le résultat sur l'homologie effective des produits tensoriels (corollaire 2.1.12) fournissent une formule de Künneth (géometrique) en homologie effective : étant donné deux ensembles simpliciaux $F$ et $B$ à homologie effective, on en déduit une homologie effective du produit $F \times B$. Cela peut être interprété comme un théorème "à la Serre" dans le cas des fibrés triviaux. Dans le paragraphe suivant on décrira la suite spectrale de Serre en homologie effective pour des fibrés quelconques.

La méthode à employer pour faire le passage entre les fibrés triviaux et les fibrés généraux est, bien sûr, le Lemme de Perturbation, d'où l'intérêt de définir des filtrations canoniques sur les complexes $C(F \times B)$ et $C(F) \otimes C(B)$.

Un générateur $(x_n, y_n)$ de $C(F \times B)$ est de degré filtrant plus petit ou égal à $q$ s'il existe $\bar{y}_q \in B_q$ tel que $y_n = s_{i_{n-q}} \ldots s_{i_1} \bar{y}_q$. D'ailleurs, $C(F) \otimes C(B)$ est filtré par les dimensions de la base :

$$\{C(F) \otimes C(B)\}^{(m)} := \bigoplus_{q \leq m} C(F) \otimes C_q(B).$$

La démonstration du lemme suivant est une simple vérification :

**Lemme 3.1.4** *Les opérateurs d'Alexander-Whitney, d'Eilenberg-MacLane et de Shih préservent les filtrations canoniques qu'on vient d'introduire.*

## 3.2 Le théorème d'Eilenberg-Zilber tordu et le théorème de Brown.

On donne dans ce paragraphe une démonstration élémentaire (due à Shih [33] ; voir aussi [10], [17]) du théorème d'Eilenberg-Zilber tordu. Ce théorème établit une équivalence d'homotopie (mieux, une réduction) entre le complexe de chaînes de l'espace total d'une fibration et un certain produit tensoriel "tordu" (il faudrait mieux dire "perturbé") des complexes de chaînes de la fibre et de la base. Ce résultat est fréquemment considéré comme équivalent au théorème de Brown sur l'existence des cochaînes de torsion (qui fournit comme sous-produit une équivalence d'homotopie du même genre). Les deux théorèmes sont en fait d'une nature assez différente. Par exemple, le théorème d'Eilenberg-Zilber tordu admet une démonstration dans un cadre plux général que celui de Brown. Une deuxième différence plus profonde (et aussi plus importante et subtile) sera détaillée dans le paragraphe suivant.

La notion de groupe structural de fibré ne joue aucun rôle dans la démonstration du théorème d'Eilenberg-Zilber tordu à l'aide de méthodes de perturbation. Ceci permet donc d'énoncer ce résultat dans le cadre des fibrations sans groupe structural tel qu'elles ont été originalement introduites dans [5]. Malheureusement, on ne dispose pas d'une terminologie bien établie pour ce type de fibrations : les termes "produit cartésien tordu", "fibré simplicial", "fibration simpliciale", ... ont des significations différentes selon les auteurs et les articles. Les noms employés ici pour les différents types de fibrés restent donc un peu arbitraires.

**Définition 3.2.1** *([5])*

*Soient $F$, $B$ deux ensembles simpliciaux. Un* produit cartésien tordu *sur $F \times B$ est défini par des applications ensemblistes $\kappa : F_n \times B_n \to F_{n-1}$, $n > 0$, vérifiant :*

$$\begin{aligned}
&\kappa(\partial_1 f, \partial_1 b) = \kappa(\kappa(f,b), \partial_0 b) \ , \\
&\partial_i \kappa(f,b) = \kappa(\partial_{i+1} f, \partial_{i+1} b) \ \textit{si } i > 0 \\
&s_i \kappa(f,b) = \kappa(s_{i+1} f, s_{i+1} b) \quad \textit{et} \\
&\kappa(f, s_0 b) = \partial_0 f.
\end{aligned}$$

Un produit cartésien tordu $\{F \times B, \kappa\}$ définit un "espace total" $F \times_\kappa B$

d'une fibration, de fibre $F$ et de base $B$, défini par :

$$\begin{array}{lcl} (F \times_\kappa B)_n & := & F_n \times B_n \ , \\ \partial_i(f,b) & := & (\partial_i f, \partial_i b) \ \text{ si } \ i > 0 \ , \\ \partial_0(f,b) & := & (\kappa(f,b), \partial_0 b) \ \text{ et} \\ s_i(f,b) & := & (s_i f, s_i b). \end{array}$$

Remarquer que les simplexes dégénérés de $F \times_\kappa B$ et de $F \times B$ sont les mêmes. Ainsi les deux complexes de chaînes $C(F \times_\kappa B)$ et $C(F \times B)$ sont égaux comme modules gradués. La seule différence entre les opérateurs de bord de $C(F \times_\kappa B)$ et de $C(F \times B)$ provient des faces d'indice zéro ; celles de $F \times_\kappa B$ seront notées $\partial_0^\kappa$.

On voit que le complexe de chaînes $C(F \times_\kappa B)$ est obtenu à partir de $C(F \times B)$ par la perturbation $\partial_0^\kappa - \partial_0$. Cette perturbation a de bonnes propriétés par rapport à la filtration canonique de $C(F \times B)$ :

**Lemme 3.2.2** *Le morphisme de modules gradués $\partial_0^\kappa - \partial_0$ diminue l'indice de filtration dans $C(F \times B)$.*

**Démonstration.**

Soit $(x_n, y_n)$ un générateur de $C_n(F \times B)$ de degré filtrant $q$ ; il existe donc $\bar{y}_q \in B_q$ tel que $y_n = s_{i_{n-q}} \ldots s_{i_1} \bar{y}_q$. On considère deux cas :

**i)** $0 \in \{i_1, \ldots, i_{n-q}\}$. Alors $y_n = s_0 \bar{y}_{n-1}$ où $\bar{y}_{n-1} \in B_{n-1}$. D'où : $\partial_0^\kappa(x_n, y_n) = \partial_0^\kappa(x_n, s_0\bar{y}_{n-1}) = (\kappa(x_n, s_0\bar{y}_{n-1}), \partial_0 s_0 \bar{y}_{n-1}) = (\partial_0 x_n, \partial_0 y_n) = \partial_0(x_n, y_n)$. Ainsi $(\partial_0^\kappa - \partial_0)(x_n, y_n) = 0$ et l'indice de filtration a donc été diminué.

**ii)** $0 \notin \{i_1, \ldots, i_{n-q}\}$. Alors $\partial_0 y_n = \partial_0 s_{i_{n-q}} \ldots s_{i_1} \bar{y}_q = s_{i_{n-q}-1} \ldots s_{i_1 - 1} \partial_0 \bar{y}_q$, d'où $\partial_0^\kappa(x_n, y_n)$ et $\partial_0(x_n, y_n)$ appartient tous les deux à $C(F \times B)^{(q-1)}$.

Compte tenu du lemme précédent et du résultat 3.1.4, on peut perturber la réduction d'Eilenberg-Zilber pour obtenir :

**Théorème 3.2.3 Théorème d'Eilenberg-Zilber tordu**

*Si $F \times_\kappa B$ est un produit cartésien tordu, alors le complexe de chaînes $C(F \times_\kappa B)$ est homotopiquement équivalent à un produit tensoriel "tordu" $C(F) \otimes_\kappa C(B)$. Plus précisement, un algorithme permet de construire une* réduction *: $\{C(F \times_\kappa B), C(F) \otimes_\kappa C(B), AW^\kappa_{F,B}, EML^\kappa_{F,B}, SHI^\kappa_{F,B}\}$.*

**Algorithme 3.2.4**
Entrée : *un produit cartésien tordu* $F \times_\kappa B$ ;
Sortie : *une réduction* $\{C(F \times_\kappa B), C(F) \otimes_\kappa C(B), AW^\kappa_{F,B}, EML^\kappa_{F,B}, SHI^\kappa_{F,B}\}$.

La méthode de perturbation a été créée par Shih [33] pour donner précisement cette démonstration du théorème d'Eilenberg-Zilber tordu. C'est aussi pour formaliser et systématiser cette démonstration que R. Brown [10], puis Gugenheim [17] ont introduit la version plus générale du Lemme de Perturbation énoncée dans 2.2.3.

Pour énoncer le théorème de Brown, on doit se placer dans le cadre des fibrations simpliciales à groupe structural.

**Définitions 3.2.5** *Soit $G$ un groupe simplicial qui agit* à droite *sur un ensemble simplicial $F$ (l'action $F \times G \to F$ est un morphisme simplicial).*

*Soit $B$ un autre ensemble simplicial ; une* application de torsion *$\tau : B_* \to G_{*-1}$ est une famille de fonctions $\tau : B_n \to G_{n-1}$, $n > 0$, vérifiant :*

$$\begin{aligned} \partial_0\tau(b) &= \tau(\partial_1 b)\tau(\partial_0 b)^{-1} , \\ \partial_i\tau(b) &= \tau(\partial_{i+1} b) \ \ si \ \ i > 0 , \\ s_i\tau(b) &= \tau(s_{i+1} b) \ \ et \\ \tau(s_0 b) &= \epsilon_* , \end{aligned}$$

*où $\epsilon_*$ est l'élément neutre du groupe simplicial de la dimension appropriée. Alors, cet ensemble de données $\{F, B, G,$ l'action de $G$ sur $F$, l'application de torsion $\tau\}$ définissent un* fibré simplicial *(un produit cartésien tordu* régulier, *selon la terminologie de [5]).*

Noter qu'un fibré simplicial est en particulier un produit tensoriel tordu ; dans ce cas l'espace total associé sera noté $F \times_\tau B$. Par abus de notation on employera encore le symbole $F \times_\tau B$ pour faire référence à un espace fibré.

Rappelons maintenant un certain nombre de notions :

**Définitions 3.2.6** *Étant donné un ensemble simplicial $B$, le* coproduit d'Alexander-Whitney *$\phi_B : C(B) \to C(B) \otimes C(B)$ est le composé de l'application diagonale $C(B) \to C(B \times B)$ et de l'opérateur d'Alexander-Whitney $AW_{B,B}$.*

*Si $G$ est un groupe simplicial, son* produit d'Eilenberg-MacLane $\mu_G : C(G) \otimes C(G) \to C(G)$ *est la composition de l'opérateur d'Eilenberg-MacLane $EML_{G,G}$ et du morphisme de complexes de chaînes $C(G \times G) \to C(G)$ induit par le produit de $G$.*

*Étant donné un morphisme de modules gradués $t : C(B) \to C(G)$ de* degré $-1$, *son* produit cup $t \cup t : C(B) \to C(G)$ *(degré $-2$) est le composé $t \cup t := \mu_G(t \otimes t)\phi_B$ (voir les conventions sur les signes des produits tensoriels fixées dans 2.1.11).*

*Soit maintenant $F$ un ensemble simplicial sur lequel agit (à droite) le groupe $G$ ; on en déduit un morphisme de complexes de chaînes $C(F) \otimes C(G) \to C(F)$, noté $\varphi_{F,G}$.*

*Dans ces conditions, le* produit cap *par $t$, noté $t\cap$, est le morphisme de modules gradués (degré $-1$) $C(F) \otimes C(B) \to C(F) \otimes C(B)$, donné par : $t\cap := (\varphi_{F,G} \otimes 1_{C(B)})(1_{C(F)} \otimes t \otimes 1_{C(B)})(1_{C(F)} \otimes \phi_B)$.*

*Un morphisme de modules gradués $t : C(B) \to C(G)$ de degré $-1$ est une* cochaîne de torsion *si $dt + td + t \cup t = 0$ (par extension, un morphisme de modules gradués $C(B) \to C(G)$ de degré $-1$ sera appelé* cochaîne*).*

*Rappelons encore qu'un* ensemble simplicial *est* réduit *s'il n'a qu'un seul sommet.*

Après ces préliminaires, on peut énoncer le théorème de Brown [9] :

**Théorème 3.2.7 Théorème de Brown**

*Soit $F \times_\tau B$ un fibré simplicial à groupe structural $G$ et à base $B$ réduite. Alors, il existe une cochaîne de torsion $t : C(B) \to C(G)$ telle que le produit tensoriel tordu $C(F) \otimes_t C(B)$, à différentielle $d_t := d \otimes 1 + 1 \otimes d + t\cap$, soit canoniquement homotopiquement équivalent au complexe de chaînes de l'espace total $C(F \times_\tau B)$.*

Une autre différence entre les énoncés des théorèmes d'Eilenberg-Zilber tordu et de Brown provient de que ce dernier donne plus d'informations sur la différentielle du nouveau produit tensoriel. La forme très particulière de $d_t$ permet de donner une démonstration élémentaire du théorème "suite spectrale de Serre" en homologie effective :

**Théorème 3.2.8 Théorème "suite spectrale de Serre" en homologie effective**

*Soit $F \times_\tau B$ un fibré simplicial à base réduite et sans 1-simplexe non-dégénéré. Alors si $F$ et $B$ sont à homologie effective, $F \times_\tau B$ est aussi à homologie effective.*

**Démonstration.**

D'après le théorème de Brown, il suffit de trouver une homologie effective du complexe de chaînes $C(F) \otimes_t C(B)$.

On sait, voir le corollaire 2.1.12, que $C(F) \otimes C(B)$ est à homologie effective. La différence entre $C(F) \otimes_t C(B)$ et $C(F) \otimes C(B)$ n'est autre que le produit cap $t\cap$. Il suffit donc de démontrer que la perturbation $t\cap$ est compatible (voir la définition 2.2.5 et le théorème 2.2.6) avec l'homologie effective canonique de $C(F) \otimes C(B)$. Cela résulte de deux remarques :

i) les opérateurs de l'homologie effective de $C(F) \otimes C(B)$ préservent les filtrations canoniques des produits tensoriels, sauf les homotopies qui augmentent *au plus d'une unité* l'indice de filtration.

ii) le fait que $B$ soit sans 1-simplexe non-dégénéré implique que $C_1(B) = 0$ et donc le composé $t\cap := (\varphi_{F,G} \otimes 1)(1 \otimes t \otimes 1)(1 \otimes \phi_B)$ diminue *au moins de deux unités* l'indice de filtration.

On donne dans la section suivante une version algorithmique du théorème de Brown. D'où l'algorithme décrit maintenant.

**Algorithme 3.2.9**

Entrées :

- *un fibré simplicial $F \times_\tau B$, $B$ étant réduit et sans 1-simplexe non-dégénéré ;*
- *deux ensembles simpliciaux à homologie effective $\{F, C(F), HC^1, \omega^1\}$, $\{B, C(B), HC^2, \omega^2\}$ ;*

Sortie : *un ensemble simplicial à homologie effective* $\{F \times_\tau B, C(F \times_\tau B), HC, \omega\}$.

## 3.3 Discussion.

Shih [33] a montré comment sa démonstration du théorème d'Eilenberg-Zilber tordu permet aussi d'atteindre toutes les conséquences du théorème de Brown. Il procède comme suit. Soit $F \times_\tau B$ un fibré à groupe structural $G$ et soit $d_t$ la différentielle obtenue par perturbation sur le module gradué $C(F) \otimes C(B)$ (cf. théorème 3.2.3). Soit $t : C(B) \to C(G)$ le morphisme de modules gradués de degré $-1$ (la *cochaîne*) défini par la composition $C(B) \to C(G) \otimes C(B) \to C(G) \otimes C(B) \to C(G)$, où le premier morphisme associe à $b$, élément de $C(B)$, l'élément $e_0 \otimes b$ de $C(G) \otimes C(B)$ ($e_0$ étant l'élément neutre de $G$ en dimension zéro), le deuxième est la différentielle tordue $d_t$ (provenant du fibré principal $G \times_\tau B$) et le troisième est induit par l'augmentation de $C(B)$. Shih démontre alors :

**Théorème 3.3.1 ([33])** *Étant donné un fibré simplicial $F \times_\tau B$ à groupe $G$, le morphisme $t : C(B) \to C(G)$ qu'on vient de définir est une cochaîne de torsion vérifiant : $d_t = d \otimes 1 + 1 \otimes d + t\cap$.*

On serait tenté d'affirmer que, puisque le théorème d'Eilenberg-Zilber tordu démontré "à la Shih" retrouve les mêmes propriétés que celui de Brown, les deux résultats sont équivalents. Mais Shih démontre le théorème précédent pour les formules *particulières* obtenues à l'aide de sa méthode. Se pose alors la question suivante : toute version obtenue par perturbation d'*une* réduction de type "Eilenberg-Zilber" a-t-elle ces mêmes propriétés ? Ou alternativement : le théorème d'Eilenberg-Zilber tordu est-il un résultat essentiellement unique ?

Pour répondre à cette dernière question, on peut commencer par étudier l'unicité des conditions de départ pour l'application du Lemme de Perturbation. Autrement dit, on s'intéresse à l'unicité des opérateurs dans le théorème d'Eilenberg-Zilber.

Prouté [26] a démontré que l'opérateur d'Eilenberg-MacLane est unique ; par contre l'unicité de l'opérateur d'Alexander-Whitney peut seulement être assurée si on impose de quel côté on prend les faces de "petits indices" (voir [27]). On peut ainsi introduire un autre opérateur d'Alexander-Whitney $AW^t_{F,B} : C(F \times B) \to C(F) \otimes C(B)$ ("t" pour "transposé") défini par :

$$AW^t(x_n, y_n) = \sum_{i=0}^{n} (-1)^{n+1} \partial_0 \dots \partial_{i-1} x_n \otimes \partial_{i+1} \dots \partial_n y_n.$$

Ce nouvel opérateur $AW^t$ est un autre inverse homotopique pour l'opérateur d'Eilenberg-MacLane, mais, bien entendu, l'opérateur de Shih correspondant doit être aussi modifié.

Ces différents opérateurs peuvent être repérés dans un cadre plus général où une réduction est "conjuguée" par des isomorphismes sur les complexes de chaînes (voir aussi [4]) :

**Lemme 3.3.2** *Soit $\{A, B, f, g, h\}$ une réduction est soient $\varphi_A : A \to \bar{A}$, $\varphi_B : B \to \bar{B}$ deux isomorphismes de complexes de chaînes. Alors une nouvelle réduction $\{\bar{A}, \bar{B}, \bar{f}, \bar{g}, \bar{h}\}$ est définie par :*

$$\begin{aligned} \bar{f} &:= \varphi_B f \varphi_A^{-1} , \\ \bar{g} &:= \varphi_A g \varphi_B^{-1} \quad et \\ \bar{h} &:= \varphi_A h \varphi_A^{-1} . \end{aligned}$$

Les isomorphismes qui nous intéressent dans le cas de la réduction d'Eilenberg-Zilber sont de deux sortes. D'abord les isomorphismes de "transposition" : l'image du générateur $(x, y)$ de $C(F \times B)$ est le générateur $(y, x)$ de $C(B \times F)$. Dans le cas des produits tensoriels il faut faire attention aux signes : l'image de l'élément $x_p \otimes y_q$ de $C_p(F) \otimes C_q(B)$ est $(-1)^p y_q \otimes x_p$, élément de $C_q(B) \otimes C_p(F)$. Deuxièmement les isomorphismes de "symétrisation" : les rôles des faces d'indice "petit" et celles d'indice "grand" sont renversés.

Pour rendre plus précis le sens de cet opérateur de symétrisation, on donne la définition suivante :

**Définition 3.3.3** *Soit $K$ un ensemble simplicial. On définit à partir de $K$ un nouvel ensemble simplicial, noté $K^{sy}$, tel que $K_n^{sy} := K_n$ et en dimension $n$ les opérateurs de face et dégénérescence sont : $\partial_i^{sy} := \partial_{n-i}$ et $s_i^{sy} := s_{n-i}$, respectivement.*

Les complexes de chaînes $C(K)$ et $C(K^{sy})$ sont alors isomorphes : l'application en dimension $n$ n'est autre que l'application qui à $x_n$, générateur de $C_n(K)$, fait correspondre l'élément $(-1)^{\lfloor n/2 \rfloor + 1} x_n$ de $C_n(K^{sy})$, où $\lfloor n/2 \rfloor$ est la partie entière de $n/2$. Remarquer que les ensembles simpliciaux $(F \times B)^{sy}$ et $F^{sy} \times B^{sy}$ sont canoniquement isomorphes.

Fixons maintenant deux ensembles simpliciaux $F$, $B$ et les opérateurs d'Eilenberg-Zilber de 3.1.1. Puis considérons les complexes de chaînes $C(F \times$

$B$) et $C(F) \otimes C(B)$ ; faisons la "conjugaison" de la réduction d'Eilenberg-Zilber canonique par les opérateurs de transposition et/ou de symétrisation. On construit ainsi trois réductions d'Eilenberg-Zilber :

**(i)** Celle canonique définie dans 3.1.1.
**(ii)** Celle conjuguée par transposition (ou symétrisation).
**(iii)** Celle conjuguée par le composé de l'opérateur de transposition et celui de symétrisation.

Comme on l'a déjà remarqué, l'opérateur d'Eilenberg-MacLane est toujours le même ; par contre, l'opérateur d'Alexander-Whitney correspondant à **(i)** et **(iii)** est l'opérateur habituel, tandis que dans **(ii)** on obtient $AW^t$, l'Alexander-Whitney "transposé" défini plus haut. Bien entendu les trois opérateurs de Shih sont différents.

On peut maintenant employer ces trois réductions comme des conditions de départ pour en déduire trois réductions de type "Eilenber-Zilber tordu" (voir 3.2.3).

À partir des différentielles tordues $d_t$ sur $C(F) \otimes C(B)$ qu'on obtient, il est toujours possible de définir une cochaîne $t : C(B) \to C(G)$ comme on a expliqué au début de ce paragraphe. En poussant un peu plus loin cette similitude, on pourrait naïvement croire que l'énoncé du théorème 3.3.1 reste toujours valable : la cochaîne $t$ serait de torsion et $d_t = d \otimes 1 + 1 \otimes d + t\cap$. Mais il se trouve que dans le cas **(ii)** $t$ n'est plus une cochaîne de torsion et on n'a pas d'espoir de retrouver la différentielle tordue $d_t$ à partir de la cochaîne $t$ (car $d \otimes 1 + 1 \otimes d + t\cap$ est une différentielle si et seulement si $t$ est une cochaîne de torsion).

C'est en ce sens que le théorème de Brown est plus fin que celui d'Eilenberg-Zilber tordu : certaines versions du deuxième ne produisent pas de cochaînes de torsion comme il est exigé dans le premier.

Afin de détailler un peu plus cette anomalie (qui a d'ailleurs été d'abord observée expérimentalement), on va introduire un fibré simplicial qui est "universel" dans un certain sens et qui joue un rôle essentiel dans la suite de ce travail : c'est le fibré simplicial associé à la version simpliciale de l'espace de lacets (la *construction G* de Kan [20]).

**Définition 3.3.4** *Soit $K$ un ensemble simplicial réduit (à un seul sommet). Définissons pour chaque $n \geq 0$ le groupe libre engendré par l'ensemble*

$K_{n+1} - \{s_0 x; x \in K_n\}$; *ce groupe sera noté* $G_n K$. *On définit l'application* $\tau : K_{n+1} \to G_n K$ *par :*

$\tau(x_{n+1}) = x_{n+1}$ *si* $x_{n+1} \notin \{s_0 x; x \in K_n\}$ ;
$\tau(x_{n+1}) = e_n$ ($e_n$ *étant l'élément neutre de* $G_n K$) *si* $x_{n+1} \in \{s_0 x; x \in K_n\}$.

*On munit l'ensemble gradué* $GK$ *d'une structure de groupe simplicial (libre), en définissant les opérateurs :*

$$\begin{aligned} \partial_0 \tau(x) &:= \tau(\partial_1 x)\tau(\partial_0 x)^{-1} , \\ \partial_i \tau(x) &:= \tau(\partial_{i+1} x) \quad si \quad i > 0 \ et \\ s_i \tau(x) &:= \tau(s_{i+1} x) \quad si \quad i \geq 0. \end{aligned}$$

*On a en même temps défini une application de torsion* $\tau : K \to GK$ *et donc un fibré simplicial* $GK \times_\tau K$.

Si on regarde dans ce fibré les trois réductions d'Eilenberg-Zilber tordu associés aux cas **(i)**, **(ii)** et **(iii)** évoqués plus haut, on trouve que les cas **(i)** et **(iii)** induisent bien des cochaînes de torsion : la première est, bien sûr, celle décrite par Shih [33] et la deuxième est la cochaîne explicitement définie par Szczarba [36].

Pour vérifier que la cochaîne $t : C(K) \to C(GK)$ n'est pas de torsion dans le cas **(ii)**, il suffit de calculer son expression en dimensions 1 et 2, qui est tout simplement : $t_1(x) = -e_0 + \tau(x)$, $t_2(x) = \tau(x)$. Si on particularise à l'ensemble simplicial $K = \Delta[2]$ (le 2-simplexe standard), alors un calcul direct montre que $(d_1 t_2 + t_1 d_2 + t \cup t)(0\ 1\ 2) = \tau(0\ 2)\tau(1\ 2)^{-1} + \tau(0\ 1)\tau(1\ 2) - \tau(0\ 1) - \tau(0\ 2)$, élément évidemment non-nul.

Si on s'interroge sur la raison pour laquelle la démonstration de Shih ne fonctionne plus dans ce cas, on voit qu'elle dépend fortement de la propriété suivante : l'application $1 \otimes \phi_B : C(F) \otimes C(B) \to C(F) \otimes C(B) \otimes C(B)$ ($\phi_B$ étant le coproduit d'Alexander-Whitney de $B$) reste encore un morphisme de complexes de chaînes après perturbation, $1 \otimes \phi_B : C(F) \otimes_t C(B) \to C(F) \otimes_t C(B) \otimes C(B)$. Or, la preuve de cette propriété utilise la relation essentielle $AW_{F \times_\tau B, B} = AW_{F \times B, B}$ (cette égalité est évidente, car dans $AW$ la face d'indice zéro, seule différence entre $F \times_\tau B$ et $F \times B$, n'est pas concernée du côté gauche). Mais, cette relation n'est plus vraie si on remplace $AW$ par $AW^t$, comme dans le cas **(ii)** de la réduction d'Eilenberg-Zilber.

D'ailleurs cette relation très étroite entre le fait que $t$ est une cochaîne de torsion et celui que $1 \otimes \phi_B : C(F) \otimes_t C(B) \to C(F) \otimes_t C(B) \otimes C(B)$ est un morphisme de complexes de chaînes n'est pas surprenante, compte tenu du résultat suivant dû à Gugenheim [17] :

**Proposition 3.3.5** *Les seules différentielles sur $C(F) \otimes C(B)$ pour lesquelles $1 \otimes \phi_B : C(F) \otimes C(B) \to C(F) \otimes C(B) \otimes C(B)$ est toujours un morphisme de complexes de chaînes sont celles provenant d'une cochaîne de torsion (via produit cap).*

Cette propriété ($1 \otimes \phi_B : C(F) \otimes_t C(B) \to C(F) \otimes_t C(B) \otimes C(B)$ est un morphisme de complexes de chaînes) est très importante pour la suite (homologie des espaces de lacets) et on rappelle qu'elle est vérifiée dans la version du théorème d'Eilenberg-Zilber tordu décrite dans le paragraphe précédent. Il est entendu que cette version sera la seule utilisée dans la suite.

# 4 Un algorithme de calcul de l'homologie des espaces de lacets itérés.

## 4.1 La construction Cobar.

La construction Cobar a été introduite par Adams [1] en 1956 pour calculer l'homologie des espaces de lacets. C'est cette construction qui est utilisée et étendue ici pour étudier le cas des espaces de lacets itérés.

Dans ce qui suit tous les produits tensoriels sont définis sur $\mathbf{Z}$ et les deux complexes $C \otimes \mathbf{Z}$, $\mathbf{Z} \otimes C$ sont canoniquement identifiés à $C$, $C$ étant un complexe de chaînes quelconque.

**Définitions 4.1.1** *Une* coalgèbre *est un ensemble de données* $\{A, \phi, \varepsilon\}$, *où* $A$ *est un complexe de chaînes et* $\phi : A \to A \otimes A$ *(nommé le* coproduit*),* $\varepsilon : A \to \mathbf{Z}$ *(la* counité*) sont des morphismes de complexes de chaînes vérifiant* $(1_A \otimes \varepsilon)\phi = 1_A$, $(\varepsilon \otimes 1_A)\phi = 1_A$ *et* $(1_A \otimes \phi)\phi = (\phi \otimes 1_A)\phi$.

*Un* $A$-comodule à droite *est un ensemble de données* $\{X, \phi_X\}$, *où* $X$ *est un complexe de chaînes et* $\phi_X : X \to X \otimes A$ *(le* coproduit extérieur*) est un morphisme de complexes de chaînes tel que* $(1_X \otimes \varepsilon)\phi_X = 1_X$ *et* $(1_X \otimes \phi)\phi_X = (\phi_X \otimes 1_A)\phi_X$.

*On définit de même la notion de* $A$*-comodule à gauche.*

*Une coalgèbre* $A$ *est* connexe *si* $A_0 = \mathbf{Z}$ *et si* $\varepsilon \mid A_0 : A_0 \to \mathbf{Z}$ *est l'identité.* $A$ *est* simplement connexe *si elle est connexe et si* $A_1 = 0$.

*Étant donné une coalgèbre connexe* $A$*, on définit un nouveau complexe de chaînes* $\bar{A}$ *comme suit :*

$$\bar{A}_n = \begin{cases} 0 & si\ n = 0 \\ A_n & si\ n > 0. \end{cases}$$

*On peut alors définir un morphisme de complexes de chaînes* $\bar{\phi} : \bar{A} \to \bar{A} \otimes \bar{A}$ *par* $\bar{\phi}(\bar{a}) := \phi(\bar{a}) - 1 \otimes \bar{a} - \bar{a} \otimes 1$. *Si* $\{X, \phi_X\}$ *est un* $A$*-comodule à droite, on définit un morphisme de complexes de chaînes* $\bar{\phi}_X : X \to X \otimes \bar{A}$ *où* $\bar{\phi}_X(x) := \phi_X(x) - x \otimes 1$ *(de même pour un* $A$*-comodule à gauche). Ces "coproduits"* $\bar{\phi}$, $\bar{\phi}_X$ *sont encore associatifs.*

*Le produit tensoriel* $\bar{A} \otimes \ldots \otimes \bar{A}$ *de* $r$ *exemplaires de* $\bar{A}$ *sera noté* $\bar{A}^r$.

**Définition 4.1.2** *Soient $A$ une coalgèbre simplement connexe, $X$ un $A$-comodule à droite et $Y$ un $A$-comodule à gauche. Le complexe $Cobar^A(X,Y)$ est défini comme la totalisation du bicomplexe :*

$$X \otimes Y \stackrel{\psi_0}{\to} X \otimes \bar{A} \otimes Y \stackrel{\psi_1}{\to} X \otimes \bar{A}^2 \otimes Y \to \ldots,$$

*où*

$$\begin{aligned}\psi_n(x \otimes \bar{a}_1 \otimes \ldots \otimes \bar{a}_n \otimes y) &= \bar{\phi}_X(x) \otimes \bar{a}_1 \otimes \ldots \otimes \bar{a}_n \otimes y + \\ &+ \sum_{i=1}^{n}(-1)^i x \otimes \ldots \otimes \bar{\phi}(\bar{a}_i) \otimes \ldots \otimes y + \\ &+ (-1)^{n+1} x \otimes \bar{a}_1 \otimes \ldots \otimes \bar{a}_n \otimes \bar{\phi}_Y(y).\end{aligned}$$

Rappelons (voir définition 2.3.1) que dans un bicomplexe de ce type, l'indice de colonne est compté négativement dans la bigraduation. Ainsi le degré dans le complexe $Cobar^A(X,Y)$ d'un élément $x \otimes \bar{a}_1 \otimes \ldots \otimes \bar{a}_n \otimes y$ du groupe $X_r \otimes A_{k_1} \otimes \ldots \otimes A_{k_n} \otimes Y_s$ est $r + k_1 + \ldots + k_n + s - n$.

Si les complexes de chaînes $A$, $X$ et $Y$ sont à homologie effective, la version homologie effective du théorème de Künneth 2.1.12 assure que chaque colonne du bicomplexe précédent est à homologie effective. Puisque $A$ est simplement connexe, le bicomplexe est simple et la proposition 2.3.5 s'applique.

**Proposition 4.1.3** *Soient $A$ une coalgèbre simplement connexe, $X$ un $A$-comodule à droite et $Y$ un $A$-comodule à gauche. Si $A$, $X$, $Y$ sont des complexes de chaînes à homologie effective, alors $Cobar^A(X,Y)$ est un complexe de chaînes à homologie effective.*

**Algorithme 4.1.4**

Entrées :

- *une coalgèbre simplement connexe $A$ ;*
- *un $A$-comodule à droite $X$ ;*
- *un $A$-comodule à gauche $Y$ ;*
- *trois complexes à homologie effective $\{A, HA, \omega^1\}$, $\{X, HX, \omega^2\}$, $\{Y, HY, \omega^3\}$ ;*

Sortie : *un complexe à homologie effective $\{Cobar^A(X,Y), HC, \omega\}$.*

Le bicomplexe $Cobar^A(X,Y)$ est canoniquement isomorphe à l'un des bicomplexes utilisés par Eilenberg et Moore pour définir le foncteur Cotor, introduit dans [14]. Il en résulte le corollaire suivant.

**Corollaire 4.1.5** *Soient $A$ une coalgèbre simplement connexe, $X$ un $A$-comodule à droite et $Y$ un $A$-comodule à gauche. Si $A$, $X$, $Y$ sont des complexes de chaînes à homologie effective, alors le $\mathbf{Z}$-module gradué $Cotor^A(X,Y)$ est calculable.*

**Algorithme 4.1.6**
Entrées :

- *une coalgèbre simplement connexe $A$ ;*
- *un $A$-comodule à droite $X$ ;*
- *un $A$-comodule à gauche $Y$ ;*
- *trois complexes à homologie effective $\{A, HA, \omega^1\}$, $\{X, HX, \omega^2\}$, $\{Y, HY, \omega^3\}$ ;*
- *un entier positif $n$ ;*

Sortie : *le groupe $Cotor_n^A(X,Y)$.*

Étudions maintenant un cas particulier de la construction Cobar. Une coalgèbre $A$ peut être considerée comme un $A$-comodule à droite (ou à gauche) de façon évidente. Alors le complexe $Cobar^A(A,\mathbf{Z})$ est acyclique avec des opérateurs d'homotopie explicites $h_n : A \otimes \bar{A}^n \to A \otimes \bar{A}^{n-1}$ définis par $h_n = \varepsilon \otimes i \otimes 1_{\bar{A}^{(n-1)}}$, où $n \geq 1$ et $i : \bar{A} \hookrightarrow A$ est l'inclusion canonique.

Ce bicomplexe n'est autre que la résolution Cobar acyclique (non-réduite) de la coalgèbre $A$. Remarquons qu'on a ainsi construit une réduction explicite de $Cobar^A(A,\mathbf{Z})$ vers le complexe trivial $\mathbf{Z}$. On applique maintenant à cette réduction le foncteur de tensorisation à gauche (sur $\mathbf{Z}$) par un complexe de chaînes $C$ ; on obtient :

**Proposition 4.1.7** *Soit $A$ une coalgèbre simplement connexe et $C$ un complexe de chaînes. Si on munit $C \otimes A$ de la structure de $A$-comodule extension (de coproduit $1_C \otimes \phi$), alors une réduction canonique de $Cobar^A(C \otimes A, \mathbf{Z})$ sur $C$ est définie.*

**Algorithme 4.1.8**
Entrées :

- *une coalgèbre simplement connexe* $A$ ;
- *un complexe de chaînes* $C$ ;

Sortie : *une réduction* $\{Cobar^A(C \otimes A, \mathbf{Z}), C, f, g, h\}$.

## 4.2 Homologie effective des espaces fibrés.

Le but de ce paragraphe est de démontrer un théorème de type "Eilenberg-Moore" en homologie effective, comme on a fait dans le chapitre précédent pour la suite spectrale de Serre. Une méthode analogue est utilisée : on résout d'abord le cas des fibrés triviaux (à l'aide de la proposition 4.1.7) et ensuite, par perturbation, le cas des fibrés quelconques est traité.

On sait que, pour tout ensemble simplicial $B$, le complexe $C(B)$ admet une structure canonique de coalgèbre ; la counité est l'augmentation et le coproduit est l'application d'Alexander-Whitney $\phi_B : C(B) \to C(B) \otimes C(B)$ (voir la définition en 3.2.6). La coalgèbre $C(B)$ est simplement connexe si et seulement si $B$ est *réduit* (autrement dit a un seul sommet) et n'a pas de 1-simplexe non-dégénéré.

Désormais tous les fibrés sont à base réduite et sans 1-simplexe non-dégénéré.

Considérons d'abord un fibré trivial $F \times B$, autrement dit le produit cartésien de deux ensembles simpliciaux. Si on suppose que $F \times B$ est à homologie effective, alors on en déduit (théorème d'Eilenberg-Zilber) que le complexe $C(F) \otimes C(B)$ est aussi à homologie effective. Si on munit maintenant $C(F) \otimes C(B)$ de la structure de $C(B)$-comodule extension, la proposition 4.1.7 s'applique et fournit une réduction de $Cobar^{C(B)}(C(F) \otimes C(B), \mathbf{Z})$ sur $C(F)$. Notons cette réduction $\{Cobar^{C(B)}(C(F) \otimes C(B), \mathbf{Z}), C(F), f, g, h\}$.

Donc si $C(F \times B)$ et $C(B)$ sont à homologie effective, on a là un algorithme de calcul de l'homologie effective de la fibre $F$ (par application de la proposition 4.1.3) ; c'est une propriété de division : connaissant l'homologie du produit et celle d'un facteur, on détermine l'homologie de l'autre facteur.

**Algorithme 4.2.1**

Entrées :

- *un ensemble simplicial $F$ ;*
- *deux ensembles simpliciaux à homologie effective $\{F \times B, C(F \times B), HC^1, \omega^1\}$, $\{B, C(B), HC^2, \omega^2\}$, $B$ étant réduit et sans 1-simplexe non-dégénéré ;*

Sortie : *un ensemble simplicial à homologie effective $\{F, C(F), HC, \omega\}$.*

Par perturbation homologique, on va ensuite résoudre le cas des fibrés non triviaux.

On définit d'abord une filtration canonique du complexe $Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z})$. On dira qu'un générateur est de degré filtrant plus petit ou égal à $m$ s'il appartient à $C(F)\otimes C_{r_0}(B)\otimes C_{r_1}(B)\otimes\ldots\otimes C_{r_n}(B)$, où $r_0+r_1+\ldots+r_n-n\leq m$. Puisque $r_1,\ldots,r_n$ doivent être strictement plus grands que 1, on voit que le seul élément de degré $-1$ est l'élément neutre. Remarquons que l'homotopie $h$ de la réduction $\{Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z}),C(F),f,g,h\}$ augmente au plus d'une unité le degré de filtration.

Soit maintenant $F\times_\tau B$ un fibré simplicial. Le théorème d'Eilenberg-Zilber tordu (cf. 3.2.3) fournit une réduction de $C(F\times_\tau B)$ sur $C(F)\otimes_t C(B)$. Comme remarqué dans le chapitre précédent, la différentielle du produit tordu $C(F)\otimes_t C(B)$ est à son tour le résultat d'une perturbation de la différentielle du produit tensoriel banal par un produit cap, noté $t\cap$, d'une cochaîne de torsion (voir le théorème 3.3.1).

Reprenons le bicomplexe $Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z})$ ; sa $n$-ième colonne est $C(F)\otimes C(B)\otimes\overline{C(B)}^n$. On perturbe maintenant la différentielle de chaque colonne pour la remplacer par le complexe $C(F)\otimes_t C(B)\otimes\overline{C(B)}^n$. De cette façon on a construit un morphisme de modules gradués $Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z})\rightarrow Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z})$, de degré -1, noté $t\cap\otimes 1\otimes\ldots\otimes 1$. Ce morphisme de modules gradués satisfait deux propriétés, réunies dans le lemme suivant.

**Lemme 4.2.2** **(i)** *Le morphisme $t\cap\otimes 1\otimes\ldots\otimes 1$ définit bien une perturbation (autrement dit, si on ajoute ce morphisme à la différentielle de $Cobar^{C(B)}(C(F)\otimes C(B),\mathbf{Z})$, on trouve encore une différentielle). De plus, le complexe obtenu par perturbation est exactement le complexe $Cobar^{C(B)}(C(F)\otimes_t C(B),\mathbf{Z})$, où le complexe $C(F)\otimes_t C(B)$ est muni d'une structure de $C(B)$-comodule à droite par le coproduit extérieur : $1\otimes\phi_B$ : $C(F)\otimes_t C(B)\rightarrow C(F)\otimes_t C(B)\otimes C(B)$.*

**(ii)** *Le morphisme $t\cap\otimes 1\otimes\ldots\otimes 1$ diminue au moins de* deux *unités l'indice de filtration dans $Cobar^{C(B)}(C(F)\otimes_t C(B),\mathbf{Z})$.*

**Démonstration.**

**(i)** Cette propriété est évidente une fois démontré le fait que $1\otimes\phi_B$ : $C(F)\otimes_t C(B)\rightarrow C(F)\otimes_t C(B)\otimes C(B)$ est un morphisme de complexes de

chaînes ; mais c'est justement ce qui résulte de la version choisie du théorème d'Eilenberg-Zilber tordu (voir la proposition 3.3.5 et les remarques finales du paragraphe **3.3**).

**(ii)** L'argument est le même que celui de la démonstration du théorème de Serre en homologie effective (théorème 3.2.8). Noter encore que la base $B$ est supposée réduite et sans 1-simplexe non-dégénéré.

Il résulte de cette étude que la réduction $\{Cobar^{C(B)}(C(F) \otimes C(B), \mathbf{Z}), C(F), f, g, h\}$ peut être perturbée par $t \cap \otimes 1 \otimes \ldots \otimes 1$. Un fait remarquable dans cette application du lemme de perturbation est que la différentielle de $C(F)$ n'est pas modifiée pendant le processus de perturbation :

**Proposition 4.2.3** *Si on applique le lemme de perturbation à la réduction* $\{Cobar^{C(B)}(C(F) \otimes C(B), \mathbf{Z}), C(F), f, g, h\}$ *et la perturbation* $t \cap \otimes 1 \otimes \ldots \otimes 1$, *on obtient une autre réduction* $\{Cobar^{C(B)}(C(F) \otimes_t C(B), \mathbf{Z}), C(F), f_\infty, g_\infty, h_\infty\}$, *où la différentielle de* $C(F)$ *n'a pas été modifiée.*

**Démonstration.**

Soit $\rho = t \cap \otimes 1 \otimes \ldots \otimes 1$ et reprenons les formules du Lemme de Perturbation (théorème 2.2.3) : $d_\infty := d_{C(F)} + f\rho\Sigma_\infty g$. Il suffit de démontrer que $f\rho\Sigma_\infty g = 0$. Pour ce faire, remarquer que $g : C(F) \to Cobar^{C(B)}(C(F) \otimes_t C(B), \mathbf{Z})$ est tout simplement l'inclusion canonique $C_n(F) \to C_n(F) \otimes C_0(B)$ ($B$ est réduit ; voir la proposition 4.1.7). Mais $t\cap$ est nul sur $C_n(F) \otimes C_0(B)$ (car la cochaîne de torsion est de source $C(B)$ et degré $-1$), d'où $f\rho\Sigma_\infty g = 0$.

Remarquer aussi que $g_\infty = g$, l'inclusion canonique.

**Algorithme 4.2.4**

Entrées :

- *la réduction* $\{Cobar^{C(B)}(C(F) \otimes C(B), \mathbf{Z}), C(F), f, g, h\}$
- *la perturbation* $t \cap \otimes 1 \otimes \ldots \otimes 1$

Sortie : *une réduction* $\{Cobar^{C(B)}(C(F) \otimes_t C(B), \mathbf{Z}), C(F), f_\infty, g, h_\infty\}$

De cette façon l'homologie effective de la fibre peut être calculée par l'intermédiaire d'une construction Cobar et le même raisonnement que dans le cas du fibré trivial peut être utilisé :

**Théorème 4.2.5** *Dans un fibré simplicial à base réduite et à 1-squelette trivial, si la base et l'espace total sont à homologie effective, alors la fibre est à homologie effective.*

**Algorithme 4.2.6**

Entrées :

- *un fibré simplicial* $F \times_\tau B$, $B$ *étant réduit et sans 1-simplexe non-dégénéré.*
- *deux ensembles simpliciaux à homologie effective* $\{F \times_\tau B, C(F \times_\tau B), HC^1, \omega^1\}$, $\{B, C(B), HC^2, \omega^2\}$

Sortie : *un ensemble simplicial à homologie effective* $\{F, C(F), HC, \omega\}$

Ce théorème est la version en homologie effective du résultat principal d'Eilenberg et Moore ([14], théorème 12.2, page 220). Dans [17] Gugenheim a aussi étudié la suite spectrale d'Eilenberg-Moore à l'aide du Lemme de Perturbation, mais sans aboutir à la détermination de l'équivalence d'homotopie $\omega$, essentielle pour l'itération.

## 4.3 Les espaces de lacets.

Dans ce paragraphe on étudie le cas particulier des fibrés associés aux espaces de lacets. On calcule ici l'homologie effective de l'espace total d'un tel fibré (calculer l'homologie effective de cet espace est équivalent à trouver une homotopie de contraction sur son complexe de chaînes, car cet espace total est contractile). Ceci et le théorème 4.2.5 qu'on vient de démontrer sur les espaces fibrés quelconques, nous permet de calculer l'homologie de l'espace de lacets en fonction de l'homologie de l'espace de départ.

Rappelons (définition 3.3.4) que, étant donné un ensemble simplicial réduit $K$, son espace de lacets sera pour nous la construction $G$ de Kan [20] ; autrement dit, l'espace de lacets de $K$ est le groupe simplicial libre $GK$, engendré en dimension $n$ par l'ensemble $K_{n+1} - \{s_0 x; x \in K_n\}$. L'application canonique $\tau : K_{*+1} \to G_*K$ fournit une structure de fibré simplicial, dont l'espace total est noté $GK \times_\tau K$.

Un ensemble simplicial réduit $K$ étant fixé, on notera $E := GK \times_\tau K$ l'espace total et on travaillera sur le complexe de chaînes *non-normalisé* de $E$, noté ici $\bar{C}(E)$. L'homotopie de contraction de ce complexe qu'on utilisera est celle définie par May dans [24].

On introduit d'abord quelques notations. Soit $k_0$ le seul sommet de $K$. On notera $k_n$ la $n$-ième dégénérescence de ce sommet $k_0$ ; autrement dit, $k_n := s_0^n k_0$. Ainsi $k_n$ est un simplexe de dimension $n$ de l'ensemble simplicial $K$.

Maintenant, étant donné un élément $g_n$ de $G_nK$ et un simplexe $x_{n+1}$, élément de $K_{n+1}$, on définit un élément $[g_n, x_{n+1}]$ du groupe $\bar{C}_n(E)$ par la formule :

$$[g_n, x_{n+1}] := (g_n \tau(x_{n+1}), \partial_0 x_{n+1}) - (g_n, k_n).$$

Remarquer que si le simplexe $x_{n+1}$ est une dégénérescence du point base $k_0$, l'élément de $\bar{C}_n(E)$ qu'on vient de définir est nul ; autrement dit, $[g_n, k_{n+1}] = 0$.

On note $B_n$ la famille constituée par les éléments $[g_n, x_{n+1}]$ où on enlève les cas dégénérés correspondants à $x_{n+1} = k_{n+1}$ ; c'est-à-dire, on définit :

$$B_n := \{[g_n, x_{n+1}]; x_{n+1} \neq k_{n+1}\}.$$

Notons enfin $e_n$ l'élément neutre du groupe $G_nK$.

Le lemme suivant est démontré par May ([24], page 120) :

**Lemme 4.3.1** *$B_n \bigcup \{(\epsilon_n, k_n)\}$ est une base de $\bar{C}_n(E)$.*

On vérifie aisément l'égalité suivante, essentielle pour la suite :

$$\partial_i[g,x] = [\partial_i g, \partial_{i+1} x] \tag{1}$$

On définit, suivant May, un opérateur $S_n : \bar{C}_n(E) \to \bar{C}_{n+1}(E)$ par les formules :

$$\begin{array}{lcl} S_n[g,x] & := & \sum_{i=0}^{n} (-1)^i [s_i g, s_0^{i+1} \partial_1^i x] \text{ et} \\ S_n(\epsilon_n, k_n) & := & (\epsilon_{n+1}, k_{n+1}). \end{array}$$

En appliquant les égalités 1, on vérifie :

**Lemme 4.3.2 (May [24])** *$S$ définit une homotopie de contraction sur $\bar{C}(E)$ (autrement dit, $dS + Sd = 1$).*

De même, en utilisant les égalités 1, on obtient :

**Lemme 4.3.3** *$S_{n+1} S_n [g,x] = 0$, pour tout $[g,x] \in B_n$.*

**Remarque 4.3.4** *Soient $f : \bar{C}(E) \to \mathbf{Z}$, $g : \mathbf{Z} \to \bar{C}(E)$, l'augmentation et la coaugmentation, respectivement. Ce sont des morphismes de complexes de chaînes si on considère le groupe $\mathbf{Z}$ comme un complexe de chaînes concentré en dimension zéro. Il est facile de vérifier que l'ensemble de données $\{\bar{C}(E), \mathbf{Z}, f, g, S\}$ définit une réduction.*

Il s'agit maintenant de montrer que les mêmes formules fournissent une homotopie de contraction sur le complexe de chaînes *normalisé* $C(E)$. Il suffit de montrer que l'image par $S$ d'un simplexe dégénéré est incluse dans le sous-complexe de $\bar{C}(E)$ engendré par les simplexes dégénérés. Pour ce faire, on va d'abord exprimer $S$ dans la base canonique de $\bar{C}(E)$. Notons $\bar{S}_n$ le composé de la contraction $S_n : \bar{C}_n(E) \to \bar{C}_{n+1}(E)$ et de la projection canonique $\bar{C}_{n+1}(E) \to C_{n+1}(E)$. Ce nouvel opérateur peut être défini par récurrence comme suit :

$$\bar{S}_n(e_n, k_n) := 0$$

et si $(g_n, x_n) \neq (e_n, k_n)$, on applique les formules suivantes :

**(A)** $(e_n, x_n) = (e_n, k_n) + [e_n, s_0 x_n]$

**(B)** si $g_n = g'_n \tau(y_{n+1})$ :

$(g_n, x_n) = (g'_n, k_n) + [g'_n, y_{n+1}] - [g_n, s_0 \partial_0 y_{n+1}] + [g_n, s_0 x_n]$

**(C)** si $g_n = g'_n \tau(y_{n+1})^{-1}$ :

$(g_n, x_n) = (g'_n, k_n) - [g_n, y_{n+1}] + [g'_n, s_0 \partial_0 y_{n+1}] + [g_n, s_0 x_n]$.

**Lemme 4.3.5**

$$\bar{S}_n[g_n, x_{n+1}] = \sum_{i=0}^{n} (-1)^i (s_i g_n, s_0^i \partial_1^i x_{n+1}).$$

**Démonstration.**

Rappelons la définition de l'opérateur d'homotopie $S$ :

$$S_n[g_n, x_{n+1}] := \sum_{i=0}^{n} (-1)^i [s_i g_n, s_0^{i+1} \partial_1^i x_{n+1}]$$

Il suffit donc pour démontrer le lemme de vérifier que la différence entre les éléments $[s_i g_n, s_0^{i+1} \partial_1^i x_{n+1}]$ et $(s_i g_n, s_0^i \partial_1^i x_{n+1})$ de $\bar{C}_{n+1}(E)$ appartient au sous-groupe engendré par les $(n+1)$-simplexes dégénérés de $E$. Pour ce faire, noter que par définition :

$$[s_i g_n, s_0^{i+1} \partial_1^i x_{n+1}] = (s_i g_n . \tau(s_0^{i+1} \partial_1^i x_{n+1}), \partial_0 s_0^{i+1} \partial_1^i x_{n+1}) - (s_i g_n, k_{n+1}).$$

Maintenant, puisque $\tau(s_0^{i+1} \partial_1^i x_{n+1}) = e_{n+1}$, on trouve :

$$[s_i g_n, s_0^{i+1} \partial_1^i x_{n+1}] = (s_i g_n, s_0^i \partial_1^i x_{n+1}) - (s_i g_n, k_{n+1}).$$

Cette égalité preuve le lemme, car l'élément $(s_i g_n, k_{n+1})$ est un simplexe dégénéré de $E$.

**Lemme 4.3.6** $\bar{S}_n(s_i g_{n-1}, x_n) = \bar{S}_n[s_i g_{n-1}, s_0 x_n]$.

**Démonstration.**

La preuve sera faite par récurrence sur la longueur du mot $g_{n-1}$, élément du groupe libre $G_{n-1}K$.

On commence par le cas trivial (longueur zéro) où $g_{n-1} = e_{n-1}$. Puisque $[s_i e_{n-1}, s_0 x_n] = (e_n, x_n) - (e_n, k_n)$, on obtient la suite d'égalités suivante : $\bar{S}_n(s_i e_{n-1}, x_n) = \bar{S}_n(e_n, x_n) = \bar{S}_n(e_n, k_n) + \bar{S}_n[s_i e_{n-1}, s_0 x_n] = \bar{S}_n[s_i e_{n-1}, s_0 x_n]$.

Supposons maintenant que $g_{n-1}$ a une longueur supérieure ou égale à 1. Deux cas sont alors possibles : ou bien $g_{n-1} = g'_{n-1}\tau(y_n)$ ou bien $g_{n-1} = g'_{n-1}\tau(y_n)^{-1}$ (où $y_n$ est un $n$-simplexe de $K$). On ne traite ici que le premier cas, compte tenu que la démonstration du deuxième cas est analogue.

Soit donc $g_{n-1} = g'_{n-1}\tau(y_n)$, d'où $s_i g_{n-1} = s_i g'_{n-1}\tau(s_{i+1}y_n)$. Si on applique alors à l'élément $(s_i g_{n-1}, x_n)$ les formules données plus haut, on trouve :

$$(s_i g_{n-1}, x_n) = (s_i g'_{n-1}, k_n) + [s_i g'_{n-1}, s_{i+1}y_n] - [s_i g_{n-1}, s_0\partial_0 s_{i+1}y_n] + [s_i g_{n-1}, s_0 x_n].$$

Et en appliquant l'opérateur $\bar{S}$ :

$$\begin{aligned} \bar{S}_n(s_i g_{n-1}, x_n) \;=\; & \bar{S}_n(s_i g'_{n-1}, k_n) + \\ & \bar{S}_n[s_i g'_{n-1}, s_{i+1}y_n] - \\ & \bar{S}_n[s_i g_{n-1}, s_0\partial_0 s_{i+1}y_n] + \\ & \bar{S}_n[s_i g_{n-1}, s_0 x_n]\ . \end{aligned}$$

Il suffit de démontrer que, dans la dernière formule, les trois premiers expressions à droite sont nulles. Par récurrence on obtient $\bar{S}_n(s_i g'_{n-1}, k_n) = 0$. Pour les deux termes suivants, on applique le lemme 4.3.5 et on trouve :

$$\bar{S}_n[s_i g'_{n-1}, s_{i+1}y_n] = \sum_{j=0}^{n} (-1)^j (s_j s_i g'_{n-1}, s_0^j \partial_1^j s_{i+1}y_n)$$

et

$$\bar{S}_n[s_i g_{n-1}, s_0\partial_0 s_{i+1}y_n] = \sum_{j=0}^{n} (-1)^j (s_j s_i g_{n-1}, s_0^j \partial_1^j s_0\partial_0 s_{i+1}y_n)\ .$$

Un calcul direct montre que tous les simplexes du type $(s_j s_i g'_{n-1}, s_0^j \partial_1^j s_{i+1}y_n)$ ou $(s_j s_i g_{n-1}, s_0^j \partial_1^j s_0\partial_0 s_{i+1}y_n)$ sont dégénérés. Cette remarque finit le pas de récurrence et donc la preuve du lemme.

Remarquer que pour les simplexes du type $(s_i g_{n-1}, x_n)$, l'expression de $\bar{S}_n$ est plutôt simple :

$$\bar{S}_n(s_i g_{n-1}, x_n) = \sum_{j=1}^{n} (-1)^j (s_j s_i g_n, s_0^j \partial_1^{j-1} x_n)$$

Il est important dans cette dernière formule que l'indice $j$ commence en $j = 1$ et non en $j = 0$. Cela permet de donner une démonstration directe du lemme principal :

**Lemme 4.3.7** *Si $(g_n, x_n)$ est dégénéré, alors $\bar{S}_n(g_n, x_n) = 0$.*

Ce dernier lemme implique que $S$ est compatible avec la relation de dégénérescence et on peut donc définir une homotopie de contraction $H$ sur le complexe de chaînes normalisé $C(E)$. Des lemmes 4.3.3 et 4.3.7, il résulte aussi que $H^2 = 0$. Autrement dit, on a construit une réduction $\{C(GK \times_\tau K), \mathbf{Z}, f, g, H\}$, où $f$ et $g$ sont l'augmentation et la coaugmentation, respectivement. Ou encore d'une autre façon :

**Théorème 4.3.8 (May)** *Si $K$ est un ensemble simplicial réduit, alors l'espace total du fibré en espaces de lacets $GK \times_\tau K$ est à homologie effective.*

**Algorithme 4.3.9**

Entrée : *un ensemble simplicial réduit $K$*

Sortie : *un ensemble simplicial à homologie effective* $\{GK \times_\tau K, C(GK \times_\tau K), \omega\}$

D'où, en appliquant le théorème 4.2.5 :

**Corollaire 4.3.10** *Si $K$ est un ensemble réduit, sans 1-simplexe non-dégénéré et à homologie effective, alors son espace de lacets $GK$ est à homologie effective.*

**Algorithme 4.3.11**

Entrée : *un ensemble simplicial à homologie effective* $\{K, C(K), HC, \omega\}$, *$K$ étant réduit et sans 1-simplexe non-dégénéré ;*

Sortie : *un ensemble simplicial à homologie effective* $\{GK, C(GK), HC^1, \omega^1\}$.

Maintenant le calcul de l'homologie des espaces de lacets itérés est une simple conséquence de ce résultat général. Remarquons seulement que les problèmes évoqués dans le premier chapitre (proposition 2.3.2) sur la simplicité des complexes qui apparaissent dans l'homologie effective sont automatiquement résolus dans le cas des espaces de lacets pour l'homologie

effective canoniquement construite. Plus précisement, si un ensemble simplicial est muni d'une homologie effective *n-simple* (c'est-à-dire, où tous les complexes de chaînes sont nuls pour les degrés $0 < m < n$), alors l'homologie effective de son espace de lacets qu'on vient de construire est $(n-1)$-simple. En particulier, si la donnée de départ est un ensemble simplicial *fini* et à $n$-squelette trivial, on peut itérer le procédé sans jamais utiliser la proposition 2.3.2.

**Corollaire 4.3.12** *Si $K$ est un ensemble simplicial réduit, sans $m$-simplexe non-dégénéré ($1 \leq m \leq n$ , $n \geq 1$), et à homologie effective, alors son espace de lacets itéré $G^nK$ est à homologie effective.*

**Algorithme 4.3.13**

Entrées :

- *un nombre entier $n \geq 1$ ;*
- *un ensemble simplicial à homologie effective $\{K, C(K), HC, \omega\}$, où $K$ est à $n$-squelette trivial ;*

Sortie : *un ensemble simplicial à homologie effective* $\{G^nK, C(G^nK), HC^n, \omega^n\}$.

# 5 Les calculs sur machine.

## 5.1 Le logiciel.

Les algorithmes décrits dans les chapitres précédents ont été implantés sur machine sous forme d'un ensemble de programmes Lisp. Une première version (5000 lignes de texte Common Lisp) a été terminée en février 90 ; elle était écrite avec comme premier objectif une démonstration de faisabilité d'une réalisation concrète des méthodes de l'homologie effective ; d'une part l'essai était concluant, d'autre part l'expérience ainsi acquise justifiait aussitôt l'écriture d'une autre version tenant compte des nombreux défauts constatés après coup dans la première version. Le gain de rapidité d'exécution de la deuxième version par rapport à la première est en général de l'ordre de 10. Noter aussi que le progrès purement matériel a entraîné parallèlement un gain supplémentaire d'un autre facteur 10 ! Ainsi par exemple le calcul de $H_7(\Omega^2(Moore(\mathbf{Z}_2, 4)))$ (où $Moore(\mathbf{Z}_2, 4)$ est l'espace dont les seuls groupes d'homologie non nuls se trouvent en dimension zéro, $\mathbf{Z}$, et en dimension quatre, $\mathbf{Z}_2$) qui nécessitait 44 heures avec la première version sur la machine disponible en 90 (SUN-3-60) est maintenant exécuté en 10 minutes avec notre deuxième version sur un SUN-4-490.

Cependant les algorithmes utilisés ont une complexité évidemment exponentielle et si on veut donc calculer $H_p(\Omega^q X)$ il est nécessaire d'être raisonnable quant au choix de $p$, $q$ et $X$. Ceci dit, le problème de trouver une version optimale de ces algorithmes reste toujours ouvert et pose des questions intéressantes de complexité.

Le logiciel est constitué des modules suivants :

1. CC. Complexes de chaînes de $\mathbf{Z}$-modules libres.
2. CCEH. Complexes de chaînes à homologie effective et perturbation homologique.
3. HOMOLOGY-GROUPS. Calcul des groupes d'homologie ordinaires d'un complexe de chaînes effectif (donc de type fini en chaque degré).
4. TPR. Produits tensoriels de complexes de chaînes.
5. SS. Ensembles simpliciaux.
6. EZ. Le théorème d'Eilenberg-Zilber.
7. TWPR. Le produit cartésien tordu associé à un espace de lacets et le théorème d'Eilenberg-Zilber tordu.

## 8. EILENBERG-MOORE. La suite spectrale.

Les algorithmes implantés suivent, à quelques détails près, le plan décrit dans le texte qui précède, si bien qu'un lecteur Common Lisp (voir [35]) pourrait lire les listings sans difficulté majeure. Seule différence notable : n'ont été écrits que les algorithmes particuliers menant à la solution du problème de l'homologie des espaces de lacets. En particulier, les algorithmes sur les bicomplexes ou sur les fibrés quelconques n'ont pas été programmés en toute généralité ; seulement les cas nécessaires pour le calcul des groupes d'homologie des espaces de lacets ont été traités. Ces généralisations sont de simples exercices de programmation.

Quant aux types de données employés, chaque *classe*, suivant la terminologie du chapitre **1**, a été implantée à l'aide de l'outil `defstruct` de Common Lisp [35]. A titre d'example on décrit l'implémentation faite pour les complexes de chaînes (comparer avec la description donnée sur machine *théorique* en **1.3**) et les morphismes de complexes de chaînes.

Un objet de type `CC` (complexe de chaînes) aura les champs suivants :

**1)** `EQC`. Une fonction de test de l'égalité de deux générateurs (on rappelle que les complexes de chaînes considérés sont des **Z**-modules *libres à base distinguée*).

**2)** `CBS`. Un champ permettant éventuellement (cas effectif) d'obtenir les informations nécessaires sur les bases des **Z**-modules du complexe de chaînes. Les informations contenues dans ce champ peuvent être de deux sortes : ou bien le mot clé `:locally-effective` ou bien une fonction qui fait correspondre à un entier $n$ la liste des générateurs en ce degré.

**3)** `D`. L'opérateur de bord. Il est codé comme un "morphisme" de complexes de chaînes, structure qu'on décrit plus loin.

**4)** `IDN`. Un numéro d'identification ("plaque d'immatriculation" sans signification mathématique).

**5)** `ORG`. Un champ documentaire d'information sur l'origine du complexe de chaînes.

Examinons maintenant l'une des structures les plus riches du logiciel : celle de morphisme de complexes de chaînes (`MRP`). Dans cette structure on code non seulement les morphismes de complexes de chaînes au sens strict

(compatibles avec les opérateurs de bord), mais plus généralement des morphismes de modules gradués : y compris les opérateurs de bord eux-mêmes, les opérateurs d'homotopie, ...

Voici la liste de champs d'un objet de type MRP :

**1)** SRC. Un objet de type CC (complexe de chaînes) qui est la source du morphisme.

**2)** TRG. Le but du morphisme.

Remarquer qu'il y a une certaine mutuelle recursivité entre les types CC et MRP. Un objet de type CC contient une composante de type MRP (l'opérateur de bord), tandis qu'un morphisme contient deux composantes de type CC (sa source et son but).

**3)** DGR. Le degré du morphisme (il sera nul dans le cas banal, $+1$ dans le cas des opérateurs d'homotopie, $-1$ dans le cas des opérateurs de bord).

**4)** STR (stratégie). Ce champ contient le mot clé :gnr ou bien le mot clé :cmb. L'interprétation du champ suivant (F) dépend de la valeur du champ STR ; y voir donc un aiguillage CASE d'un RECORD de Pascal.

**5)** F. Si le champ STR contient le mot clé :gnr (pour *générateur*), le champ F contient une fonction travaillant sur tout *générateur* du complexe source et retournant la combinaison image. Si au contraire le champ STR contient le mot clé :cmb (pour *combinaison*), alors le champ F contient une fonction travaillant directement sur toute combinaison élément du complexe source et retournant une combinaison élément du complexe résultat. La justification de cette façon de faire est expliquée plus loin.

**6)** IDN.

**7)** ORG.

Ces deux champs jouent le même rôle que dans les complexes de chaînes.

**8)** ???-NCALL. Un compteur pour enregistrer le nombre de fois que le morphisme a été sollicité pour travailler sur des combinaisons.

**9)** ?-NCALL. Analogue au champ qui précède, mais pour les appels sur des générateurs individuels.

**10)** RSL. Dans ce champ on se donne la possibilité de garder en mémoire un résultat quand le morphisme a travaillé sur un générateur, de façon à pouvoir utiliser ce résultat si le morphisme est appelé ultérieurement pour le même argument.

C'est là l'une des différences essentielles entre la version commentée du logiciel et la toute première. Dans la première version les fonctions recalcu-

laient fréquemment sur les mêmes arguments et le temps de calcul en était évidemment augmenté. Cette impression a été confirmée par le gain de rapidité constaté entre les deux versions du logiciel (voir l'exemple donné au début de ce paragraphe).

Pourtant il peut se faire que ce mécanisme de mise en mémoire des résultats ne soit pas toujours favorable. La recherche dans les tables pour vérifier si un générateur a été déjà interrogé n'est pas gratuite en temps de calcul. Ceci peut même être catastrophique dans le cas des morphismes "simples" fréquemment utilisés dans le logiciel : il est par exemple inutile de garder en mémoire les résultats des morphismes nuls, des morphismes identités, etc. D'où l'intérêt de la possibilité de choisir la stratégie selon laquelle va travailler un morphisme : par combinaisons (`:cmb`) si on ne veut pas conserver les résultats, par générateurs (`:gnr`) dans les autres cas.

La première version du logiciel souffrait aussi d'un autre défaut. Aucun renseignement n'était accessible sur les parties du logiciel les plus utilisées, celles qui nécessitaient le plus de temps de calcul, etc. Ceci nous empêchait notamment de connaître les fonctions sur lesquelles un effort d'optimisation est souhaitable. Pour corriger cette situation, un certain nombre de champs documentaires a été ajouté aux diverses structures (par exemple les champs `IDN` et `ORG` dans `CC` ou les champs 6 à 9 de `MRP`) qui permettent de faire une étude statistique après d'exécution. De plus, les résultats mis en mémoire dans le champ `RSL` de chaque morphisme conservent le temps utilisé pour calculer l'image de chaque générateur et le nombre de fois que ce générateur a été sollicité par le morphisme envisagé.

Un exemple de l'utilisation de ces outils techniques et d'une séance de travail avec ce logiciel est décrit dans l'annonce [30], article qui est donné en annexe à la fin de ce mémoire.

## 5.2 Exemples et vérifications.

Le logiciel a déjà travaillé sur un ensemble significatif d'exemples, notamment sur des espaces de lacets itérés deux et trois fois. On étudie dans ce paragraphe l'homologie de certains espaces pour lesquels il a pu être démontré que les résultats trouvés sur machine concordent avec des résultats obtenus par ailleurs. Dans d'autres cas on n'a pas su vérifier entièrement les résultats (ce qui évidemment justifie l'intérêt du logiciel ...), mais il a néanmoins été démontré qu'ils ne contredisaient jamais d'autres résultats théoriques tels qu'un sur homologie connue à coefficients dans un corps, par exemple.

Dans ce paragraphe et le suivant le foncteur "espace de lacets" sera noté $\Omega$ et le foncteur "suspension" $S$. L'espace de Moore de type $(\pi, n)$ (autrement dit, l'espace dont les seuls groupes d'homologie non-nuls se trouvent en dimension zéro, $\mathbf{Z}$, et en dimension $n$, $\pi$) sera noté $M(\pi, n)$.

Les premiers calculs effectués par le logiciel ont été, bien sûr, des vérifications dans le cas facile du premier espace de lacets. Par exemple, le logiciel calcule en moins d'une minute de temps d'exécution le groupe $H_{30}(\Omega M(\mathbf{Z}_2, 4)) = (\mathbf{Z}_2)^{64}$. Pour faire ce calcul le programme a dû diagonaliser une matrice de nombres entiers à $113 \times 144$ éléments.

Pour l'homologie de l'espace $\Omega M(\mathbf{Z}_2, 2)$ on a constaté expérimentalement qu'en dimension $n$ il y avait autant de facteurs de $\mathbf{Z}_2$ que d'unités dans le $n$-ième nombre de la suite de Fibonacci.

Ceci peut être aisément démontré en appliquant le résultat suivant de Moore (inspiré semble-t-il d'un résultat antérieur de G. Whitehead).

**Théorème 5.2.1 (Moore [25])** *Soit $X$ un espace connexe et soit $n > 0$. Alors $H_n(\Omega SX)$ est isomorphe à la somme directe de $\sum_{r+s=n, r>0} H_r(X) \otimes H_s(\Omega SX)$ et de $\sum_{r+s=n-1} Tor(H_r(X), H_s(\Omega SX))$.*

Le premier calcul significatif accompli par le logiciel a été celui de l'homologie (jusqu'à la dimension 7) de $\Omega^2 S^3$. Comme on le verra, le nombre de résultats théoriques nécessaires pour déterminer l'homologie entière de $\Omega^2 S^3$ est déjà non négligeable.

**Proposition 5.2.2** *Les huit premiers groupes d'homologie de $\Omega^2 S^3$ sont $\mathbf{Z}$, $\mathbf{Z}$, $\mathbf{Z}_2$, $\mathbf{Z}_2$, $\mathbf{Z}_2 \oplus \mathbf{Z}_3$, $\mathbf{Z}_2 \oplus \mathbf{Z}_3$, $(\mathbf{Z}_2)^2$ et $(\mathbf{Z}_2)^2$.*

Commençons par énoncer les résultats à utiliser pour démontrer cette proposition. Les deux premiers sont dus au travail de plusieurs auteurs (Araki-Kudo, Browder, Dyer-Lashof, Nishida, Cohen-Lada-May), mais ils ont été extraits de l'article [11] de Cohen.

Définissons une suite d'entiers $I = (i_1, \ldots, i_j)$ comme *admissible* si $0 < i_1 \leq \ldots \leq i_j$ ; notons $\lambda(I) = i_j$. On sait que $H_n(\Omega^k S^{n+k}) \cong \mathbf{Z}$ ; soit $x_n$ un générateur de ce groupe d'homologie. Des opérations homologiques $Q_i$ peuvent être définies $Q_i : H_q(\Omega^n X) \to H_{2q+i}(\Omega^n X)$, où $0 \leq i \leq n-1$. Si $I = (i_1, \ldots, i_j)$ est une suite admissible, on notera $Q_I$ le composé $Q_{i_1} \ldots Q_{i_j}$.

**Théorème 5.2.3** *Il existe un isomorphisme d'algèbres de Hopf* $H_*(\Omega^k S^{n+k}; \mathbf{Z}_2) \cong \mathbf{Z}_2[Q_I x_n]$, *où* $I$ *est admissible,* $\lambda(I) \leq k-1$ *et* $Q_I x_n$ *est primitif.*

**Théorème 5.2.4** *Soit* $p$ *un nombre premier impair. Alors, l'homologie à coefficients dans* $\mathbf{Z}_p$ *de* $\Omega^2 S^3$ *est isomorphe comme algèbre de Hopf à*

$$\bigotimes_{i \geq 0} \bigwedge [y_{2p^i-1}] \bigotimes_{j \geq 1} \mathbf{Z}_p[x_{2p^j-2}] \ ,$$

*où* $y_r$, $x_s$ *ont pour degrés* $r$ *et* $s$, *respectivement.*

**Théorème 5.2.5 (Cohen [12])** *Si l'homologie d'un espace* $X$ *n'a pas de p-torsion, alors la p-torsion de* $H_*(\Omega^2 S^2 X; \mathbf{Z})$ *est toujours d'ordre* $p$.

**Démonstration de la proposition 5.2.2**

On calcule d'abord (en utilisant les théorèmes 5.2.3 et 5.2.4) l'homologie à coefficients dans $\mathbf{Z}_p$ de $\Omega^2 S^3$. On applique ensuite la formule de coefficients universels :

$$H_n(X; G) \cong (H_n(X) \otimes G) \oplus Tor(H_{n-1}(X); G)$$

et le théorème 5.2.5 pour déterminer l'homologie entière.

On commence par la $\mathbf{Z}_2$-torsion. Soit $x_1$ le générateur du groupe $H_1(\Omega^2 S^3; \mathbf{Z}_2) \cong \mathbf{Z}_2$. On applique le théorème 5.2.3, où $k = 2$, $n = 1$. Donc $\lambda(I) \leq 1$ et les seules suites admissibles sont $(1, \ldots, 1)$. On voit que les seuls générateurs élémentaires sont $x_1 \in H_1(\Omega^2 S^3; \mathbf{Z}_2)$, $y_3 := Q_1 x_1 \in H_3(\Omega^2 S^3; \mathbf{Z}_2)$ et $y_7 := Q_1 y_3 \in H_7(\Omega^2 S^3; \mathbf{Z}_2)$.

On obtient ainsi :

$$
\begin{array}{lll}
H_2(\Omega^2 S^3;\mathbf{Z}_2) \cong \mathbf{Z}_2 ; & & \text{engendré par } (x_1)^2 . \\
H_3(\Omega^2 S^3;\mathbf{Z}_2) \cong \mathbf{Z}_2 \oplus \mathbf{Z}_2 ; & & (x_1)^3 ,\ y_3 = Q_1 x_1 . \\
H_4(\Omega^2 S^3;\mathbf{Z}_2) \cong \mathbf{Z}_2 \oplus \mathbf{Z}_2 ; & & (x_1)^4 ,\ x_1.y_3 . \\
H_5(\Omega^2 S^3;\mathbf{Z}_2) \cong \mathbf{Z}_2 \oplus \mathbf{Z}_2 ; & & (x_1)^5 ,\ (x_1)^2.y_3 . \\
H_6(\Omega^2 S^3;\mathbf{Z}_2) \cong \mathbf{Z}_2 \oplus \mathbf{Z}_2 \oplus \mathbf{Z}_2 ; & & (x_1)^6 ,\ (y_3)^2 ,\ (x_1)^3.y_3 . \\
H_7(\Omega^2 S^3;\mathbf{Z}_2) \cong (\mathbf{Z}_2)^4 ; & (x_1)^7 , & x_1.(y_3)^2 ,\ (x_1)^4.y_3 ,\ y_7 = Q_1 y_3 .
\end{array}
$$

Etudions maintenant la $\mathbf{Z}_3$-homologie. Dans le théorème 5.2.4, si $p = 3$ on voit que la liste de générateurs pour l'algèbre extérieure commence par : $y_1, y_5, y_{17}, \ldots$ , tandis que celle de l'algèbre de polynômes est : $x_4, x_{16}, \ldots$ . D'où :

$$
\begin{array}{lll}
H_1(\Omega^2 S^3;\mathbf{Z}_3) \cong \mathbf{Z}_3 ; & \text{engendré par } y_1 . \\
H_2(\Omega^2 S^3;\mathbf{Z}_3) \cong 0 . & \\
H_3(\Omega^2 S^3;\mathbf{Z}_3) \cong 0 . & \\
H_4(\Omega^2 S^3;\mathbf{Z}_3) \cong \mathbf{Z}_3 ; & \text{engendré par } x_4 . \\
H_5(\Omega^2 S^3;\mathbf{Z}_3) \cong \mathbf{Z}_3 \oplus \mathbf{Z}_3 ; & y_1 \otimes x_4 ,\ y_5 . \\
H_6(\Omega^2 S^3;\mathbf{Z}_3) \cong \mathbf{Z}_3 ; & y_1 \otimes y_5 . \\
H_7(\Omega^2 S^3;\mathbf{Z}_3) \cong 0 . &
\end{array}
$$

Le même raisonnement montre que l'homologie à coefficients dans $\mathbf{Z}_p$ ($p$ premier plus grand que 3) commence au-delà de la dimension 7 (sauf les cas triviaux en dimensions 0 et 1). Cela prouve en même temps que les groupes d'homologie entière de $\Omega^2 S^3$ en dimensions 2, 3, 4, 5, 6 et 7 sont finis (utiliser coefficients universels).

La conclusion de ces calculs est que les groupes $H_i(\Omega^2 S^3)$, $2 \leq i \leq 7$ sont de torsion et ont seulement de la 2 et de la 3 torsion. Mais, puisque l'homologie de $S^1$ est libre, le théorème 5.2.5 de Cohen assure qu'il peut y avoir seulement des composantes $\mathbf{Z}_2$ et/ou $\mathbf{Z}_3$ ; donc un raisonnement pour les dimensions sur $\mathbf{Z}_2$ et $\mathbf{Z}_3$ suffira pour déterminer $H_i(\Omega^2 S^3)$, $i = 2,3,4,5,6,7$.

En appliquant récursivement le théorème des coefficients universels au groupe $G = \mathbf{Z}_2$, on trouve $H_i(\Omega^2 S^3) \otimes \mathbf{Z}_2 \cong \mathbf{Z}_2$ si $i = 2,3,4,5$ et $H_i(\Omega^2 S^3) \otimes \mathbf{Z}_2 \cong (\mathbf{Z}_2)^2$ si $i = 6,7$.

Encore une fois, grâce aux coefficients universels, on calcule $H_i(\Omega^2 S^3) \otimes \mathbf{Z}_3 \cong 0$ si $i = 2,3,6,7$ et $H_4(\Omega^2 S^3) \otimes \mathbf{Z}_3 \cong H_5(\Omega^2 S^3) \otimes \mathbf{Z}_3 \cong \mathbf{Z}_3$ , ce qui

termine la démonstration de la proposition.

Dans cette vérification, le théorème de Cohen nous permet de faire un raisonnement sur le nombre de composantes isomorphes à $\mathbf{Z}_p$. Or, ce théorème ne s'applique plus si l'homologie de l'espace de départ a de la $p$-torsion et, surtout, il ne dit rien pour les espaces de lacets itérés trois fois (même si l'espace est à son tour une suspension itérée trois fois ...). On retrouvera ces problèmes plus loin, mais on va d'abord faire quelques vérifications plus faciles :

**Proposition 5.2.6** *Les cinq premiers groupes d'homologie de l'espace* $\Omega S\Omega M(\mathbf{Z}_2, 2)$ *sont* $\mathbf{Z}$, $\mathbf{Z}_2$, $(\mathbf{Z}_2)^2$, $(\mathbf{Z}_2)^6$ *et* $(\mathbf{Z}_2)^{16}$.

**Démonstration.**
Il suffit de remarquer que $M(\mathbf{Z}_2, 2)$ a le même type d'homotopie que la suspension du plan projectif réel et d'appliquer patiemment deux fois le théorème 5.2.1 de Moore ...

On considère maintenant le cas de l'espace de lacets d'un bouquet de deux espaces $X_1$ et $X_2$. Une première approche pour calculer l'homologie entière de $\Omega(X_1 \vee X_2)$ est d'utiliser le résultat de Hilton et Milnor qui démontre l'existence d'une équivalence d'homotopie entre $\Omega(X_1 \vee X_2)$ et l'espace :

$$(\Omega X_1) \times (\Omega X_2) \times \Omega S((\Omega X_1) \wedge (\Omega X_2)).$$

Si on connait l'homologie entière de $\Omega X_1$, $\Omega X_2$ et $(\Omega X_1) \wedge (\Omega X_2)$, en appliquant la formule de Künneth à ce dernier espace et le théorème 5.2.1 de Moore, on pourrait déterminer l'homologie entière de $\Omega(X_1 \vee X_2)$.

Une deuxième méthode (Aguadé-Castellet [3], Katz [21]) pour calculer l'homologie entière de $\Omega(X_1 \vee X_2)$ consiste à exprimer $H_n(\Omega(X_1 \vee X_2))$ en termes de $H(\Omega X_1)$ et $H(\Omega X_2)$ en utilisant une formule explicite. Pour ce faire, Aguadé et Castellet définissent une famille de foncteurs $Mult_i^n$, où $n$ est un entier positif et $i$ est un entier positif ou nul plus petit que $n$. Etant donné $n$ et $i$, le foncteur $Mult_i^n$ fait correspondre à $n$ groupes abéliens $A_1, \ldots, A_n$ un autre groupe abélien $Mult_i^n(A_1, \ldots, A_n)$. Le groupe $Mult_i^n(A_1, \ldots, A_n)$ est facile à calculer, si chaque groupe $A_j$ est de type fini :

**Proposition 5.2.7 [3]**

(i) *$Mult_i^n$ est un fonteur additif.*
(ii) *$Mult_i^n$ est symétrique.*
(iii) $Mult_i^n(A_1, \ldots, A_{n-1}, \mathbf{Z}) \cong Mult_i^{n-1}(A_1, \ldots, A_{n-1})$.
(iv) $Mult_i^n(\mathbf{Z}_{r_1}, \ldots, \mathbf{Z}_{r_n}) \cong \bigoplus \begin{pmatrix} n-1 \\ i \end{pmatrix} \mathbf{Z}_{p.g.c.d.(r_1,\ldots,r_n)}$.

Le théorème principal de [3] s'énonce alors :

**Théorème 5.2.8**

$$\tilde{H}(\Omega(X_1 \vee X_2)) \cong \sum_{i_1,\ldots,i_n} \sum_{j=0}^{n-1} Mult_j^n(\tilde{H}(\Omega X_{i_1}), \ldots, \tilde{H}(\Omega X_{i_n})) ,$$

*où la première somme est définie pour les indices $(i_1, \ldots, i_n)$ égaux à $(1,2,1,2,\ldots)$ ou $(2,1,2,1,\ldots)$, $\tilde{H}$ est le foncteur d'homologie réduite et le degré de $Mult_j^n(H_{r_n}(\Omega X_{i_1}), \ldots, H_{r_n}(\Omega X_{i_n}))$ est égal à $r_1 + \cdots + r_n + j$.*

Ce résultat nous a permis de vérifier les calculs expérimentaux suivants :

**Proposition 5.2.9** *Les six premiers groupes d'homologie de l'espace $\Omega(\Omega S^3 \vee S^2)$ sont* $\mathbf{Z}$, $\mathbf{Z}^2$, $\mathbf{Z}^3 \oplus \mathbf{Z}_2$, $\mathbf{Z}^5 \oplus (\mathbf{Z}_2)^3$, $\mathbf{Z}^8 \oplus (\mathbf{Z}_2)^8 \oplus \mathbf{Z}_3$, $\mathbf{Z}^{13} \oplus (\mathbf{Z}_2)^{19} \oplus (\mathbf{Z}_3)^3$.

Considérons maintenant les résultats suivants obtenus par le logiciel :

Les six premiers groupes d'homologie de l'espace $\Omega^2 M(\mathbf{Z}_2, 3)$ sont $\mathbf{Z}$, $\mathbf{Z}_2$, $\mathbf{Z}_2$, $\mathbf{Z}_4 \oplus \mathbf{Z}_2$, $(\mathbf{Z}_2)^3$ et $(\mathbf{Z}_2)^4$.

. Remarquer que dans $H_3(\Omega^2 M(\mathbf{Z}_2, 3))$ apparaît le premier exemple de 2-torsion d'ordre 4 ; ceci n'est pas en contradiction avec le théorème 5.2.5 de Cohen, car l'homologie du plan projectif réel a de la 2-torsion.

Dans ce cas, nous n'avons pas su donner une démonstration complète de la correction des résultats, mais il a été vérifié qu'il y a cohérence du point de vue des dimensions, compte tenu de ce que, puisque $M(\mathbf{Z}_2, 3)$ est à homologie 2-primaire, il est de même pour son deuxième espace de lacets.

**Proposition 5.2.10** *Le nombre de composantes à 2-torsion des groupes* $H_i(\Omega^2 M(\mathbf{Z}_2, 3))$ *pour* $i = 1, 2, 3, 4, 5$ *sont respectivement* $1, 1, 2, 3, 4$.

La démonstration provient d'un calcul direct à l'aide d'un théorème de Browder [8] qui généralise le théorème 5.2.3 au cas d'un espace $\Omega^n S^n X$. On épargnera au lecteur les détails techniques qu'il faudrait donner pour énoncer le théorème utilisé.

Il est intéressant de remarquer que, si on suppose corrects les résultats pour l'homologie de $\Omega^2 M(\mathbf{Z}_2, 3)$, il est facile d'utiliser le théorème 5.2.8 pour vérifier les calculs suivants :

> Les six premiers groupes d'homologie de l'espace $\Omega(\Omega M(\mathbf{Z}_2, 3) \vee M(\mathbf{Z}_2, 2))$ sont $\mathbf{Z}$, $(\mathbf{Z}_2)^2$, $(\mathbf{Z}_2)^4$, $\mathbf{Z}_4 \oplus (\mathbf{Z}_2)^{11}$, $(\mathbf{Z}_2)^{32}$, $(\mathbf{Z}_2)^{87}$.

Pour calculer ce dernier groupe $H_5(\Omega(\Omega M(\mathbf{Z}_2, 3) \vee M(\mathbf{Z}_2, 2)))$ le programme a dû diagonaliser une matrice de nombres entiers à $146 \times 428$ éléments.

Le même théorème de Browder cité plus haut permet de comparer le résultat expérimental :

> Les huit premiers groupes d'homologie de l'espace $\Omega^2 M(\mathbf{Z}_2, 4)$ sont $\mathbf{Z}$, $0$, $\mathbf{Z}_2$, $0$, $\mathbf{Z}_2$, $\mathbf{Z}_4$, $(\mathbf{Z}_2)^2$, $\mathbf{Z}_2$.

à la proposition :

**Proposition 5.2.11** *Le nombre de composantes à 2-torsion des groupes* $H_i(\Omega^2 M(\mathbf{Z}_2, 4))$ *pour* $i = 2, \ldots, 7$ *sont respectivement* $1, 0, 1, 1, 2, 1$.

Dans des cas plus généraux où l'homologie des espaces a de la torsion de différentes caractéristiques et de parties libres, même le nombre de composantes d'une torsion fixée devient difficile à déterminer. Regardons par exemple l'espace $\Omega^2(M(\mathbf{Z}_2, 3) \vee S^4)$ (dont les groupes d'homologie trouvés sont $\mathbf{Z}$, $\mathbf{Z}_2$, $\mathbf{Z} \oplus \mathbf{Z}_2$, $\mathbf{Z}_4 \oplus (\mathbf{Z}_2)^2$, $\mathbf{Z} \oplus (\mathbf{Z}_2)^5$ et $\mathbf{Z} \oplus \mathbf{Z}_4 \oplus (\mathbf{Z}_2)^7$).

L'espace envisagé a le même type d'homotopie que $\Omega^2 S^2 X$, où $X$ est le bouquet du plan projectif réel et de la sphere $S^2$.

Le théorème de Browder permet alors de calculer $H_i(\Omega^2 S^2 X; \mathbf{Z}_2)$ ; ces espaces vectoriels sont de dimensions 1, 3, 4, 8 et 12, pour $i = 1, \ldots, 5$ respectivement. Maintenant, en utilisant la formule de coefficient universels

$$H_2(\Omega^2 S^2 X; \mathbf{Z}_2) \cong H_2(\Omega^2 S^2 X) \otimes \mathbf{Z}_2 \oplus Tor(H_1(\Omega^2 S^2 X); \mathbf{Z}_2),$$

on en déduit la dimension de $H_2(\Omega^2 S^2 X) \otimes \mathbf{Z}_2$ (égal à 2). Or, l'espace n'est plus 2-primaire et ce seul renseignement ne suffit pas pour déterminer le nombre de composantes à 2-torsion se trouvant dans $H_2(\Omega^2 S^2 X)$.

Considérons un dernier espace dont les premiers groupes d'homologie ont été calculés par notre logiciel. Cet espace est construit comme suit. On attache un disque de dimension trois $D^3$ à l'espace $\Omega S^3$ par une application d'attachement $f : S^2 \to \Omega S^3$ de degré deux telle que $\pi_2(\Omega S^3 \bigcup_f D^3) = \mathbf{Z}_2$. Maintenant, l'espace $\Omega S^3 \bigcup_f D^3$ est à homologie effective et simplement connexe. On peut donc faire calculer à notre logiciel des groupes d'homologie de l'espace de lacets $\Omega(\Omega S^3 \bigcup_f D^3)$. Les résultats obtenus sont :

> Les six premiers groupes d'homologie de l'espace $\Omega(\Omega S^3 \bigcup_f D^3)$ sont $\mathbf{Z}$, $\mathbf{Z}_2$, $\mathbf{Z}_2$, $\mathbf{Z} \oplus (\mathbf{Z}_2)^2$, $(\mathbf{Z}_2)^4$, $\mathbf{Z} \oplus (\mathbf{Z}_2)^6$.

Cet exemple est intéressant parce que déterminer l'effet d'un attachement cellulaire dans l'homologie d'un espace de lacets est un problème difficile qui reste notamment ouvert lorsqu'il s'agit de l'homologie à coefficients sur un corps de caractéristique deux (voir [16]).

## 5.3 Plus d'exemples.

On donne dans cette section une liste des groupes d'homologie calculés par notre logiciel.

---

$\Omega^2 S^3$.

$H_1 = \mathbf{Z}$
$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/2\mathbf{Z}$
$H_4 = \mathbf{Z}/2\mathbf{Z} \oplus \mathbf{Z}/3\mathbf{Z}$
$H_5 = \mathbf{Z}/2\mathbf{Z} \oplus \mathbf{Z}/3\mathbf{Z}$
$H_6 = (\mathbf{Z}/2\mathbf{Z})^2$
$H_7 = (\mathbf{Z}/2\mathbf{Z})^2$

---

$\Omega^3 S^4$.

$H_1 = \mathbf{Z}$
$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/2\mathbf{Z}$
$H_4 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z} \oplus \mathbf{Z}/3\mathbf{Z}$
$H_5 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z} \oplus \mathbf{Z}/3\mathbf{Z}$

---

$\Omega^2 Moore(\mathbf{Z}/2\mathbf{Z}, 3)$.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_4 = (\mathbf{Z}/2\mathbf{Z})^3$
$H_5 = (\mathbf{Z}/2\mathbf{Z})^4$

---

$\Omega^2 Moore\epsilon(\mathbf{Z}/2\mathbf{Z}, 4)$.

$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = 0$
$H_4 = \mathbf{Z}/2\mathbf{Z}$
$H_5 = \mathbf{Z}/4\mathbf{Z}$
$H_6 = (\mathbf{Z}/2\mathbf{Z})^2$
$H_7 = \mathbf{Z}/2\mathbf{Z}$

---

$\Omega^2(Moore(\mathbf{Z}/2\mathbf{Z}, 3) \bigcup_{S^3} Moore(\mathbf{Z}/2\mathbf{Z}, 3))$.

Deux espaces de Moore sont attachés suivant une sphère de dimension trois.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^2$
$H_4 = \mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^5$
$H_5 = \mathbf{Z} \oplus \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^7$

---

$\Omega^2(Moore(\mathbf{Z}/2\mathbf{Z}, 3) \vee S^4)$.

Cet espace a le même type d'homotopie que l'espace précédent.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^2$
$H_4 = \mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^5$
$H_5 = \mathbf{Z} \oplus \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^7$

---

$\Omega S\Omega Moore(\mathbf{Z}/2\mathbf{Z}, 2)$.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = (\mathbf{Z}/2\mathbf{Z})^2$
$H_3 = (\mathbf{Z}/2\mathbf{Z})^6$
$H_4 = (\mathbf{Z}/2\mathbf{Z})^{16}$

---

$\Omega(\Omega S^3 \vee S^2)$.

$H_1 = \mathbf{Z}^2$
$H_2 = \mathbf{Z}^3 \oplus \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}^5 \oplus (\mathbf{Z}/2\mathbf{Z})^3$
$H_4 = \mathbf{Z}^8 \oplus (\mathbf{Z}/2\mathbf{Z})^8 \oplus \mathbf{Z}/3\mathbf{Z}$
$H_5 = \mathbf{Z}^{13} \oplus (\mathbf{Z}/2\mathbf{Z})^{19} \oplus (\mathbf{Z}/3\mathbf{Z})^3$

---

$\Omega(\Omega Moore(\mathbf{Z}/2\mathbf{Z}, 3) \vee Moore(\mathbf{Z}/2\mathbf{Z}, 2))$.

$H_1 = (\mathbf{Z}/2\mathbf{Z})^2$
$H_2 = (\mathbf{Z}/2\mathbf{Z})^4$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^{11}$
$H_4 = (\mathbf{Z}/2\mathbf{Z})^{32}$
$H_5 = (\mathbf{Z}/2\mathbf{Z})^{87}$

---

$\Omega(\Omega S^3 \bigcup_f D^3)$.

Un disque de dimension trois est attaché à $\Omega S^3$ par une application $f : S^2 \to \Omega S^3$ de degré deux de façon que $\pi_2(\Omega S^3 \bigcup_f D^3) = \mathbf{Z}/2\mathbf{Z}$.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^2$

$H_4 = (\mathbf{Z}/2\mathbf{Z})^4$
$H_5 = \mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^6$

---

$\Omega^3(S^4 \vee S^5)$.

$H_1 = \mathbf{Z}$
$H_2 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_4 = \mathbf{Z}^2 \oplus (\mathbf{Z}/2\mathbf{Z})^2 \oplus \mathbf{Z}/3\mathbf{Z}$
$H_5 = \mathbf{Z}^3 \oplus (\mathbf{Z}/2\mathbf{Z})^3 \oplus \mathbf{Z}/3\mathbf{Z}$

---

$\Omega^3 Moore(\mathbf{Z}/2\mathbf{Z}, 4)$.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_4 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^2$
$H_5 = (\mathbf{Z}/2\mathbf{Z})^5$

---

$\Omega^3(Moore(\mathbf{Z}/2\mathbf{Z}, 4) \vee S^5)$.

$H_1 = \mathbf{Z}/2\mathbf{Z}$
$H_2 = \mathbf{Z} \oplus \mathbf{Z}/2\mathbf{Z}$
$H_3 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^2$
$H_4 = \mathbf{Z} \oplus \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^3$
$H_5 = \mathbf{Z}/4\mathbf{Z} \oplus (\mathbf{Z}/2\mathbf{Z})^9$

# References


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

[2] ADAMS J. F., HILTON P. J.- *On the chain algebra of a loop space*, Comm. Math. Helv. **30** (1956), 305-330.

[3] AGUADE J., CASTELLET M.- *The homology of* $\Omega(X \vee Y)$, Collect. Math. **29** (1978), 3-6.

[4] BARNES D. W., LAMBE L. A.- *Fixed point approach to homological perturbation theory.* Preprint.

[5] BARRAT M. G., GUGENHEIM V. K. A. M., MOORE J. C.- *On semisimplicial fibre-bundles*, Am. J. Math. **81** (1959), 639-657.

[6] BAUES H. J.-*Geometry of loop spaces and the cobar construction*, Memoirs A. M. S. **230**, 1980.

[7] BAUES H. J.- *The double bar and cobar constructions*, Comp. Math. **43** (1981), 331-341.

[8] BROWDER W.- *Homology operations and loop spaces*, Ill. J. Math. **4** (1960), 347-357.

[9] BROWN E. H.- *Twisted tensor products I*, Ann. of Math. **69** (1959), 223-246.

[10] BROWN R.- *The twisted Eilenberg-Zilber theorem*, Celebrazioni Arch. Secolo XX, Simp. Top. (1967), 34-37.

[11] COHEN F. R.- *A course in some aspects of classical homotopy theory*, Lecture Notes in Math. **1286**, 1-92.

[12] COHEN F. R.- *The homology of* $C_{n+1}$*-spaces,* $n \geq 0$, Lecture Notes in Math. **533**, 207-351.

[13] EILENBERG S., MacLANE S.- *On the groups* $H(\Pi, n)$, *I*, Ann. Math. **58** (1953), 55-106.

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

[15] EILENBERG S., ZILBER J. A.- *On products of complexes*, Am. J. Math. **75** (1959), 200-204.

[16] FELIX T., THOMAS J. C.- *Effet d'un attachement cellulaire dans l'homologie de l'espace de lacets*, Ann. Inst. Fourier **39** (1989), 207-224.

[17] GUGENHEIM V. K. A. M.- *On the chain complex of a fibration*, Ill. J. Math. **16** (1972), 398-414.

[18] GUGENHEIM V. K. A. M., LAMBE L.- *Perturbation theory in differential homological algebra I*, Ill. J. Math. **33** (1989), 566-582.

[19] HUEBSCHMAN J.- *Cohomology of metacyclic groups.* A paraître dans Trans. Amer. Math. Soc.

[20] KAN D. M.- *A combinatorial definition of homotopy groups*, Comm. Math. Helv. **67** (1958), 282-312.

[21] KATZ, E.- *A Künneth formula for coproducts of simplicial groups*, Proc. Amer. Math. Soc. **61** (1976), 117-121.

[22] LAMBE L.- *Resolutions via homological perturbation theory.* A paraître dans J. Symb. Comp.

[23] LAMBE L., STASHEFF J.- *Applications of perturbation theory of iterated fibrations*, Manuscripta Math. **58** (1987), 363-376.

[24] MAY J. P.- *Simplicial objets in Algebraic Topology*, Van Nostrand, 1967.

[25] MOORE J. C.- *Le théorème de Freudenthal, la suite exacte de James et l'invariant de Hopf généralisé*, Séminaire Henri Cartan, 1954-1955, exposé 22.

[26] PROUTE A.- *Sur la transformation d'Eilenber-MacLane*, C. R. Acad. Sc. Paris **297** (1983), 193-194.

[27] PROUTE A.- *Sur la diagonale d'Alexander-Whitney*, C. R. Acad. Sc. Paris **299** (1984), 391-392.

[28] REVESZ G.- *Lambda calculus, combinators and functional programming.*, Cambridge University Press, 1988.

[29] RUBIO J.- *Un algorithme de calcul de l'homologie des espaces de lacets itérés.* A paraître.

[30] RUBIO J., SERGERAERT F.- *Un logiciel calculant l'homologie des espaces de lacets.*

[31] SERGERAERT F.- *The computability problem in Algebraic Topology.* A paraître dans Advances in Math.

[32] SERRE J. P.- *Groupes d'homotopie et classes de groupes abéliens*, Ann. Math. **58** (1953), 258-294.

[33] SHIH W.- *Homologie des espaces fibrés*, Publ. Math. I.H.E.S. **13**, 1962.

[34] STASHEFF J.- *H-spaces from a homotopy point of view*, Lecture Notes in Math. **161**, 1970.

[35] STEELE G.- *Common Lisp, the language*, Digital Press, 1984.

[36] SZCZARBA R. H.- *The homology of twisted cartesian products*, Trans. Amer. Math. Soc. **100** (1961), 197-216.

# ANNEXE

# Un logiciel calculant l'homologie des espaces de lacets

Julio RUBIO - Francis SERGERAERT

## 1 Introduction.

Le but de cette annonce est de décrire la deuxième version d'un logiciel permettant en théorie de calculer les groupes d'homologie des espaces de lacets itérés $H_q(\Omega^p X; \mathbf{Z})$ (tout problème d'extension résolu) si $X$ est un ensemble simplicial réduit à homologie effective dont le squelette commence en dimension $p+1$. Bien entendu, compte tenu de la complexité hyperexponentielle de l'algorithme, un choix trop ambitieux pour $p$, $q$ et $X$ ne permettra pas d'obtenir un résultat en un temps raisonnable. Mais plusieurs exemples de calculs déjà faits à l'aide de ce logiciel montrent qu'il permet d'atteindre des groupes qui semblaient jusque là inaccesibles.

Le logiciel utilise les idées générales de l'homologie effective et du codage fonctionnel [8], exactement telles qu'elles ont été exploitées par le premier auteur dans l'article [7] pour démontrer la calculabilité de l'homologie des espaces de lacets itérés. Le principe en remonte à Adams (construction Cobar [1]) mais l'impossibilité d'installer une bonne structure de coalgèbre sur la construction Cobar n'autorise l'itération de la méthode qu'à homotopie près, et l'organisation du travail devient si complexe qu'elle avait semble-t-il découragé les spécialistes ; voir à ce sujet [2] et [3]. Noter que les modifications, très élémentaires, apportées aux définitions traditionnelles de l'algèbre homologique pour en faire un outil *effectif* ont donné le cadre idéal pour itérer la construction Cobar à homotopie près [7].

L'autre outil essentiel est le "Basic Perturbation Lemma"(Shih [9], R.

Brown [4], Gugenheim [5], Lambe et Stasheff [6], ...) dont certains calculs de [8] étaient en fait, sans que l'auteur le sache, des cas particuliers. Les récents développements basés autour de ce "Lemme" devraient lui valoir un nom plus à la hauteur de son importance. Rappelons que les suites exactes et suites spectrales usuelles (Serre, Eilenberg-Moore, ...) sont des *sous-produits* de ce "lemme" et que la forme très forte que Shih a donné au théorème d'Eilenberg-Zilber tordu en est aussi une conséquence.

Le logiciel est un texte Common Lisp d'environ 5000 lignes (120 Kbs à peu près) exécutable sous Lucid-Common-Lisp sur une station Sun 3/60 ; la compatibilité avec Lucid-Common-Lisp 2.0 (sur Sun 4) et avec Kyoto-Common-Lisp sur Gould et Sun a été vérifiée. La station Sun utilisée pour le développement et l'utilisation est une station de travail relativement modeste ; dans la gamme des matériels actuellement disponibles, elle se situe à peu près juste au-dessus des meilleurs PC. Une grande partie du logiciel a d'ailleurs été mise au point à l'aide de GCLisp sur un PC ordinaire. Si des calculs plus conséquents étaient jugés utiles, des moyens sans commune mesure pourraient être utilisés : le Centre d'Estudes Nucléaires de Grenoble est équipé d'un Cray-2 ; l'installation d'Allegro-Common-Lisp sur ce site est à l'étude. Un essai de nos programmes sur un Cray américan est en cours.

Les auteurs remercient chaleureusement les équipes d'informaticiens ayant réussi la conception, la mise au point et la mise à disposition du logiciel de base Common-Lisp. C'est un outil dont l'esthétique et l'efficacité suscitent notre admiration de mathématiciens. Ce logiciel a été disponible à partir de 1986, au moins en France, exactement au moment où nous en avions besoin, quelle chance !

## 2 La deuxième version.

Cette version ne diffère pas sur le fond de la précédente. Des améliorations essentiellement de deux types ont été apportées ; d'une part différentes améliorations de détail, le plus souvent relevant de techniques de programmation, ont été intégrées au logiciel ; il en est résulté un gain de temps de calcul significatif qui a permis d'aller sensiblement plus loin dans le calcul

de "nouveaux" groupes. D'autre part le logiciel garde maintenant trace de ce qu'il fait, si bien qu'il est désormais possible d'analyser son travail. On a ainsi une plus juste idée de la complication du travail d'algèbre homologique qu'il entreprend.

Diverses améliorations de détail du logiciel ont donc été introduites. Comme ce n'était pas du tout le but recherché pendant l'écriture de cette deuxième version, seules des amelíorations "évidentes" ont été apportées. Par exemple l'expérience a montré que la distinction entre *modules*, *modules gradués* et *complexes de chaînes* était plutôt nuisible. Elle vient naturellement à l'esprit d'un mathématicien dans ce genre de situation, mais il est en fait beaucoup plus efficace de considérer par exemple un module comme un complexe de chaînes dont la graduation et la différentielle n'ont pas le droit d'être utilisées sous peine d'arrêt sur erreur. Une technique très différente a été utilisée pour programmer le théorème d'Eilenberg-Zilber. Il en est résulté sur ce point du logiciel une division du temps de calcul par 100. La technique utilisée maintenant est pourtant très batarde et n'est certainement pas la meilleure.

Compte tenu de l'expérience acquise, c'était un exercice raisonnable de transformer, chaque fois que c'était possible, les versions non destructives de traitement d'objets composites (listes, structures...) par des versions destructives. Ceci a été fait sans difficulté majeure et donne probablement un gain de temps d'un facteur 2.

Une amélioration d'une nature beaucoup plus intéressante a été intégrée, consistant à tenter si possible de garder en mémoire un résultat qui risque d'être réutilisable. L'exemple détaillé plus loin indique les avantages et les limites d'une telle stratégie qui posent beaucoup de questions informatiques très intéressantes.

Avec d'innombrables autres améliorations de détail qu'il n'y a pas lieu de d'expliquer ici, le résultat suivant a été obtenu : les comparaisons de temps de calcul entre l'ancienne et la nouvelle version montrent que le nouveau temps de calcul est égal à l'ancien divisé par un facteur compris entre 3 et 60. Le plus souvent il est de l'ordre de 10. Il a ainsi été possible d'atteindre en un temps raisonnable des groupes d'homologie qui semblaient précédemment inaccessibles. Notre nouvelle version a ainsi calculé $H_5(\Omega^3(S^4 \vee S^5)) = \mathbf{Z}_2 \oplus$

$\mathbf{Z}_2 \oplus \mathbf{Z}_6 \oplus \mathbf{Z}^3$ en quatre heures, ce qui est tout-à-fait honorable sur une machine (Sun 3-60) déjà démodée malgré ses grandes qualités...

Ce résultat en gain de temps de calcul ne motivait en aucune façon l'écriture de cette deuxième version. Le but recherché était de pouvoir obtenir, une fois le calcul terminé, une idée de la nature du travail effectué. Ce but est atteint par une organisation en mémoire très soigneuse des objets manipulés, de telle façon qu'on puisse,*une fois le calcul terminé*, interroger la machine pour qu'elle donne rapidement la liste des complexes de chaînes qui ont été construits, à quelle occasion cette construction est intervenue, quels morphismes de complexes de chaînes ont été construits, combien de fois ont-ils travaillé, sur quels objets, quel temps de calcul a été nécessaire quand tel morphisme a travaillé sur tel objet,...

Ce problème n'est pas en fait complètement disjoint du précédent : tant de choses doivent être gardées en mémoire qu'il est alors tentant le cas échéant de les réutiliser ; d'où un gain évident de temps de calcul qui sera mesuré très précisément plus loin dans un cas particulier ; cependant le surcoup de gestion de ces mises en mémoire ne doit évidemment pas être négligé. Déterminer la gestion optimale en la matière est un splendide problème d'optimisation.

On détaille les résultats obtenus dans un cas particulier qui nous semble significatif ; celui du calcul du groupe $H_5(\Omega^3(S^4 \vee S^5))$. Ce calcul avec notre logiciel se présente comme suit. Dans une session Lucid-Common-Lisp, on exécutera successivement un certain nombre d'instructions Lisp. D'abord :

```
(load "Eilenberg-Moore")
```

charge en mémoire l'ensemble de notre logiciel, chargement qui demande 21 secondes. Puis :

```
(setf w (sphere-wedge 4 5))
```

repère à l'aide du symbole `w` le bouquet $S^4 \vee S^5$ construit à l'aide de la fonction `sphere-wedge` (0.2 seconde). Cet ensemble simplicial est un ensemble simplicial fini et est donc trivialement à homologie effective. Notre fonction `ess-sseh` (effective simplicial set to simplicial set with effective homology) assure la conversion, conversion qui demande 4 centièmes de seconde :

```
(setf weh (ess-sseh w))
```

Maintenant le symbole **weh** repère un objet machine qui code un ensemble simplicial à homologie effective ; un tel objet est un ensemble assez volumineux d'algorithmes capables de répondre à une multitude de questions. Notre logiciel contient une fonction `loop-space-eh` capable d'associer à un tel ensemble simplicial à homologie effective son *espace de lacets*, un nouvel ensemble simplicial à homologie effective. Il faut considérer cette fonction comme réalisant une écriture automatique des algorithmes associés à l'espace à construire (qui sera repéré par le symbole **oweh**) à partir des algorithmes associés à l'espace donné (repéré par **weh**). Ce travail d'écriture nécessite 2.6 secondes :

```
(setf oweh (loop-space-eh weh))
```

Puis on construit de même le second espace de lacets :

```
(setf ooweh (loop-space-eh oweh))
```

et le troisième :

```
(setf oooweh (loop-space-eh ooweh))
```

chacune de ces deux constructions demandant aussi 2.6 secondes.

Le logiciel a alors construit une multitude d'objets prêts à travailler au besoin, mais qui n'ont pas encore réellement travaillé. Ont été construits :

- 20 ensembles simpliciaux dont 8 effectifs et 12 localement effectifs ;
- 137 complexes de chaînes dont 103 localement effectifs ;
- 9 coalgèbres différentielles graduées dont 6 localement effectives ;
- 15 comodules différentiels gradués dont 13 localement effectifs.
- 487 morphismes de complexes de chaînes ;

- 40 réductions entre complexes de chaînes ;
- 16 équivalences d'homotopie entre complexes de chaînes.

Un objet *localement effectif* est un objet capable de répondre à des questions de nature *locale*, par exemple "Quel est le bord de tel simplexe ?" mais par contre en général incapable de répondre à une question de nature globale comme "Combien de simplexes de dimension 2 contient l'ensemble simplicial codé ?". La notion de *morphisme de complexes de chaînes* est à comprendre au sens large : un opérateur d'homotopie "est" un tel morphisme. Une *réduction* est un quintuplet $(\hat{C}, C, f, g, h)$ où $f : \hat{C} \to C$ et $g : C \to \hat{C}$ sont deux (vrais) morphismes de complexes de chaînes, $h$ est une homotopie entre $g \circ f$ et l'identité, le tout décrivant $\hat{C}$ comme somme directe de $C$ et d'un complexe acyclique. Une *équivalence d'homotopie* est une paire de réductions ayant le même grand complexe.

La table suivante indique l'origine des divers complexes de chaînes :

| | |
|---|---|
| `tpr-2cc` | **62** |
| `ss-cc` | **24** |
| `cc-bar` | **18** |
| `perturb-differential` | **18** |
| `pre-cobar-2cc` | **12** |
| `ss-cc-d` | **2** |
| `*ccz*` | **1** |

On voit que parmi ces nombreux complexes de chaînes, 62 ne sont que le produit tensoriel de deux autres complexes de chaînes. La fonction `ss-cc` construit le complexe de chaînes normalisé d'un ensemble simplicial, alors que `ss-cc-d` construit le complexe non normalisé. La fonction `cc-bar` donne "l'idéal d'augmentation" d'un complexe connexe (l'unique générateur de degré 0 est évacué) ; `*ccz*` est le complexe unité pour le produit tensoriel. La fonction `pre-cobar-2cc` nécessite deux complexes arguments, le premier une coalgèbre et le second un comodule sur cette coalgèbre ; cette fonction construit le complexe *construction cobar* de ces deux objets, à ceci près que seules les flèches verticales sont insérées. Les flèches horizontales seront considérées comme une *perturbation* de ce complexe, ce qui permet l'application

du *lemme de perturbation*, notre version de ce qui est appelé habituellement *suite spectrale*. Enfin la fonction *perturb-differential* utilise deux arguments : un complexe de chaînes et une perturbation de sa différentielle ; il construit le complexe à différentielle perturbée.

Considérons maintenant la table analogue pour les "morphismes" entre complexes de chaînes :

| | | | |
|---|---|---|---|
| build-cc | 137 | rdc-bpl-*-sigma | 9 |
| cmp-mrp | 105 | h-cpr | 6 |
| change-src-trg | 75 | pre-cobar-2rdc | 6 |
| add-mrp-to-mrp | 21 | tpr-2mrp | 6 |
| id-mrp | 19 | cobar-hor-d | 6 |
| pre-cobar-2mrp | 12 | zero-mrp | 4 |
| augmentation | 12 | aw | 3 |
| build-com | 12 | eml | 3 |
| ecop-bar | 12 | phi | 3 |
| sbt-mrp-from-mrp | 12 | dtau-d | 3 |
| build-coal | 9 | coaugmentation | 3 |
| acyc-cobar-rdc | 9 | Total | 487 |

On donne maintenant des indications succintes sur la nature de ces constructeurs de morphismes entre complexes de chaînes.

- `build-cc` : un complexe de chaînes contient une différentielle qui est (regarder !) *vraiment* un morphisme de complexe de chaînes.
- `cmp-mrp` : produit le composé de deux morphismes.
- `change-src-trg` : soit un morphisme entre deux complexes ; une perturbation de la différentielle à la source et/ou au but ne change pas le morphisme en tant que morphisme de modules gradués, mais le change en tant que morphisme de complexes. C'est une situation fréquente quand on applique le lemme de perturbation.
- `add-mrp-to-mrp` : le morphisme somme de deux morphismes.

- `id-mrp` : le morphisme identité d'un complexe de chaînes.
- `pre-cobar-2mrp` : analogue à `pre-cobar-2cc` ; la notion est foncto-rielle et s'applique donc aussi aux morphismes.
- `augmentation` : le morphisme d'augmentation sur un complexe con-nexe.
- `build-com` : un comodule contient un coproduit externe qui est un morphisme de complexes de chaînes.
- `ecop-bar` : un coproduit externe sur un sous-module d'augmentation.
- `sbt-mrp-from-mrp` : différence entre deux morphismes.
- `build-coal` : une coalgèbre contient un coproduit qui est un mor-phisme.
- `acyc-cobar-rdc` : si on prend comme comodule une coalgèbre, la con-struction cobar est acyclique, d'où une réduction de cette construction sur le complexe unité, collection de trois morphismes.
- `rdc-bpl-*-sigma` : le morphisme somme d'une série de morphismes dans le lemme de perturbation.
- `h-cpr` : l'homotopie de contraction du complexe de l'espace total d'un fibré en espaces de lacets.
- `pre-cobar-2rdc` : on peut même faire une construction "pre-cobar" de deux réductions, qui produira une réduction donc trois morphismes.
- `tpr-2mrp` : produit tensoriel de deux morphismes.
- `cobar-hor-d` : la composante horizontale de la construction cobar.
- `zero-mrp` : le morphisme nul entre deux complexes.
- `aw` : Alexander-Whitney.
- `eml` : Eilenberg-MacLane.

- `phi` : l'homotopie entre **eml** o **aw** et l'identité.
- `dtau-d` : perturbation construite par application du lemme de perturbation.
- `coaugmentation` : le morphisme de coaugmentation du complexe unité vers un complexe connexe.

Notre logiciel permet si on examine par exemple un morphisme construit par `pre-cobar-2mrp` de savoir quels étaient les deux morphismes arguments, puis d'examiner ces deux morphismes, par exemple quand et comment ils ont été construits. quels sont leurs source et but respectifs, quand et comment ces sources et buts ont été construits, etc. Cet ensemble de complexes et de morphismes est une sorte de graphe extrêmement compliqué dont les arêtes généralisées sont les constructeurs qu'on a décrits.

A ce stade déjà intéressant, le logiciel a "construit un logiciel" sous la forme de l'ensemble simplicial à homologie effective repéré par le symbole `oooweh`. C'est ce logiciel-là qu'on vient de décrire très succintement, mais il n'a toujours pas travaillé. Pour le faire travailler on peut par exemple demander l'exécution de l'instruction Lisp :

```
(homology oooweh 5)
```

et attendre patiemment quatre heures pour voir à l'écran apparaître le résultat, le cinquième groupe d'homologie de l'espace des applications continues pointées de $S^3$ vers $S^4 \vee S^5$. Pendant ces quatre heures, le logiciel va savoir exploiter simultanément trois suites spectrales d'Eilenberg-Moore, chacune étant construite sur les résultats de la précédente. Plus précisément le logiciel va construire trois **Z**-modules de type fini et deux morphismes de composé nul ; le groupe d'homologie correspondant est garanti isomorphe à celui demandé, des générateurs des classes d'homologie *dans* $\Omega^3(S^4 \vee S^5)$ *lui-même* pouvant même être fournis si c'est demandé.

Beaucoup de morphismes auront beaucoup travaillé pour ce faire. Le nombre de générateurs de complexes de chaînes concernés est 34207. La table suivante :

| | |
|---|---|
| **1** | 3767 |
| **15** | de 100 à 999 |
| **37** | de 10 à 99 |
| **8735** | de 2 à 9 |
| **25419** | 1 |

indique par exemple que l'un de ces générateurs a été interrogé par un morphisme 3767 fois, que 15 générateurs ont été respectivement interrogés par 15 morphismes un nombre de fois compris entre 100 et 999, etc. Bien entendu un même générateur peut être à la source de plusieurs morphismes différents. On voit donc que l'interrogation répétée d'un générateur par un morphisme est fréquente mais que c'est loin d'être toujours le cas. Il est intéressant aussi de savoir combien de temps a été utilisé pour calculer chaque image. D'où une table à interpréter de la même façon :

| | |
|---|---|
| **1** | 4376sec. $\simeq$ 1h13mn |
| **54** | $\geq$ 1000sec. |
| **242** | de 100 à 1000sec. |
| **833** | de 10 à 100sec. |
| **3142** | de 1 à 10sec. |
| **11821** | de 0.1 à 1sec. |
| **11672** | de 0.02 à 0.1 sec. |
| **6443** | $<$ 0.02sec. |

Le temps total est très supérieur à quatre heures ; ceci est dû à ce que de nombreux intervalles de temps sont emboîtés, par exemple quand le morphisme qui travaille est défini comme le composé de deux morphismes : le calcul va *comprendre* le calcul de l'image du même générateur par le premier morphisme, puis le calcul de l'image de plusieurs générateurs par le second morphisme.

On voit que la situation est très variée. L'une des images mises en mémoire avait nécessité plus d'une heure de calcul et la même image fut redemandée ultérieurement : une heure de calculs gagnée contre le temps ! Par contre mettre en mémoire une image qui n'a nécessité pour être calculée qu'une microseconde et qui de plus n'a jamais été redemandée est

un beau gaspillage ! Le problème d'optimisation posé là est fort complexe car il fait intervenir aussi la complexité espace. Par exemple le calcul de $H_5(\Omega^3(\text{Moore}(\mathbf{Z}_2, 4) \vee S^5))$ avait échoué avec la première version de notre logiciel pour cause de *complexité temps* : aucun problème de mémoire n'a été rencontré, mais l'évaluation grossière de la vitesse de calcul indiquait que plusieurs semaines seraient nécessaires. Avec la nouvelle version. le calcul a aussi échoué mais cette fois pour cause de *complexité espace* ; le logiciel n'était pas très loin du but mais a fini par étouffer en mémoire (si quelqu'un était vraiment intéressé par le résultat, il est évidemment accessible sur une machine un peu plus conséquente).

Il est possible d'évaluer le temps gagné par le mécanisme de mise en mémoire ; il est d'environ 5h15mn. Il faudrait soustraire de ce temps celui qui a été nécessité par la gestion beaucoup plus complexe de la mémoire, peut-être une heure.

Ces considérations sont un peu techniques, mais leur intérêt ne doit pas être sous-estimé. Le parti-pris de transformer la suite spectrale d'Eilenberg-Moore en un véritable outil algorithmique a déjà permis d'atteindre ce qui est de toute évidence la version définitive de la construction cobar, plus sophistiquée que la version originale, mais si élégante et si puissante ! Il n'est pas impossible qu'un examen plus précis de tel ou tel aspect de ce logiciel permette de découvrir d'autres trésors encore cachés.

# References


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

[2] BAUES H. J.–*Geometry of loop spaces and the cobar construction*, Memoirs A. M. S. **230**, 1980.

[3] BAUES H. J.– *The double bar and cobar constructions*, Comp. Math. – **43** (1981), 331-341.

[4] BROWN R.– *The twisted Eilenberg-Zilber theorem*, Celebrazioni Arch. Secolo XX, Simp. Top. (1967), 34-37.

[5] GUGENHEIM V. K. A. M.- *On the chain complex of a fibration*, Ill. J. Math. **16** (1972), 398-414.

[6] LAMBE L., STASHEFF J.- *Applications of perturbation theory of iterated fibrations*, Manuscripta Math. **58** (1987), 363-376.

[7] RUBIO J.- *Un algorithme de calcul de l'homologie des espaces de lacets itérés*. A paraître.

[8] SERGERAERT F.- *The computability problem in Algebraic Topology*. A paraître dans Advances in Math.

[9] SHIH W.- *Homologie des espaces fibrés*, Publ. Math. I.H.E.S. **13**, 1962.

Julio RUBIO.

Departamento de Ingeniería Eléctrica e Informática.

Facultad de Ciencias.

50009 ZARAGOZA.

SPAIN

E-address : rubio@cc.unizar.es

Francis SERGERAERT

Laboratoire de Modélisation et Calcul.

BP 53x

38041 GRENOBLE CEDEX.

FRANCE.

E-address : sergerar@imag.fr

## RÉSUMÉ

On décrit dans ce mémoire un ensemble d'algorithmes qui nous ont permis de développer un logiciel Lisp calculant l'homologie entière des espaces de lacets itérés. Dans le chapitre 1, on introduit une machine théorique (inspirée du $\lambda$-calcul) où il faut interpréter les résultats de calculabilité démontrés dans les chapitres suivants.

Le deuxième chapitre est consacré à étudier les propriétés de l'homologie effective et la théorie dite de perturbation homologique.

Le lien entre l'algèbre et la topologie, autrement dit le théorème d'Eilenberg-Zilber est traité dans le troisième chapitre. On y inclut aussi une comparaison entre le théorème d'Eilenberg-Zilber tordu et le résultat de E. Brown sur l'existence des cochaînes de torsion.

En utilisant les résultats précédents, on donne dans le chapitre 4 un algorithme de calcul de l'homologie effective des espaces de lacets itérés. Le point clé de cet algorithme est la transformation de la suite spectrale d'Eilenberg-Moore en un vrai procédé de calcul.

Enfin, le chapitre 5 est consacré à décrire quelques détails d'implémentation du logiciel et à donner une liste d'exemples des groupes d'homologie qui ont été déjà calculés sur machine.



## MOTS-CLÉS

Homologie effective, calculabilité, programmation fonctionnelle, espaces de lacets.

## MATHEMATICAL SUBJECT CLASSIFICATION

55T – 68Q – 18G.