THÈSE

PRÉSENTÉE

À L'UNIVERSITÉ DE MARNE-LA-VALLÉE

POUR OBTENIR

LE TITRE DE DOCTEUR DE
L'UNIVERSITÉ DE MARNE-LA-VALLÉE
SPÉCIALITÉ : INFORMATIQUE FONDAMENTALE

Par Sébastien Veigneau

CALCUL SYMBOLIQUE ET CALCUL DISTRIBUÉ EN COMBINATOIRE ALGÉBRIQUE

Soutenue le 10 décembre 1996, devant le jury composé de :

Alain Lascoux	*Président*
Jean-Yves Thibon	*Directeur*
Meinolf Geck	*Rapporteur*
Michel Morvan	*Rapporteur*
Piotr Pragacz	*Rapporteur*
Didier Arquès	*Examinateur*
Marie-Françoise Coste-Roy	*Examinateur*

Calcul Symbolique et Calcul Distribué en Combinatoire Algébrique

Sébastien Veigneau

Institut Gaspard Monge
Université de Marne-la-Vallée

Résumé. Ce travail est divisé en trois parties : la première présente `ACE`, un environnement de calcul en combinatoire algébrique. Cet outil dédié au système de calcul formel MAPLE regroupe plusieurs bibliothèques de fonctions spécialisées. Il permet notamment d'effectuer des calculs sur le groupe symétrique, le groupe hyperoctaédral, les fonctions symétriques, les tableaux de Young ou encore les polynômes de Schubert. Nous présentons dans un premier chapitre les différentes bibliothèques que nous avons réalisées. Le deuxième chapitre donne des exemples combinatoires illustrant les possibilités de l'environnement et le troisième chapitre présente quelques applications à la géométrie énumérative.

La deuxième partie présente nos contributions à la théorie combinatoire des polynômes de Schubert. Nous y donnons les analogues de la formule de Monk et de la formule de Pieri pour les polynômes de Schubert doubles ainsi que la formule de Pieri pour les polynômes de Schubert quantiques. Nous décrivons aussi la structure multiplicative de l'anneau des polynômes en plusieurs variables en tant que module libre de rang fini sur l'anneau des polynômes symétriques, avec comme base distinguée les polynômes de Schubert.

La dernière partie traite de problèmes liés à la distribution de calculs. Après une présentation de plusieurs expériences que nous avons menées, nous donnons un algorithme d'énumération des tableaux de rubans d'une forme et d'un poids donnés associé au calcul du polynôme de spin correspondant. Nous proposons ensuite une version distribuée de cet algorithme. Ces travaux s'inscrivent dans un programme d'expérimentation des possibilités du calcul distribué dans les domaines de la combinatoire et du calcul symbolique. Dans ce contexte, nous abordons des problèmes liés à l'étude des performances des programmes distribués en environnement `PVM` et montrons les modifications que nous avons apportées à `PGPVM` afin qu'il intègre les fonctionnalités de base d'un traceur d'exécution.

Symbolic Computation and Distributed Computation in Algebraic Combinatorics

Sébastien Veigneau

Institut Gaspard Monge
Université de Marne-la-Vallée

Abstract. This work is divided into three parts: the former introduces `ACE`, an environment devoted to computations in Algebraic Combinatorics. This tool designed for the computer algebra system MAPLE clusters several libraries of specialized functions. It allows to compute with the symmetric group, the hyperoctahedral group, symmetric functions, Young tableaux and also Schubert polynomials. In the first chapter, we give a brief description of the different libraries. The second chapter gives combinatorial examples whereas the last chapter presents some applications to enumerative geometry.

The second part exhibits our contributions to the theory of Schubert polynomials. In particular, we give the analogues of the Monk formula and the Pieri formula for double Schubert polynomials together with the Pieri formula for quantum Schubert polynomials. We also describe the multiplicative structure of the ring of polynomials in several variables regarded as a free module of finite rank over the ring of symmetric polynomials, with Schubert polynomials as a distinguished basis.

The last part deals with problems concerning distributed computations. After a short overview of several experiences we performed, we give an algorithm to enumerate all ribbon tableaux of a given shape and a given weight, associated to the computation of the corresponding spin polynomial. We thereafter propose a distributed version of this algorithm. This work makes part of an experimental study of the distributed computation possibilities in combinatorics and symbolic computation. In this context, we address some issues related to the performance analysis of programs distributed in the `PVM` environment, and we show how we modified `PGPVM` in such a way as it includes all basic features we expect from an event tracer.

Remerciements

Ce qu'on voulait faire,
c'est en le faisant qu'on le découvre.

— Alain.

Si la préparation d'une thèse nécessite un important travail personnel, elle doit tout de même beaucoup à la participation—dirais-je au dévouement, à la disponibilité, à la présence, à la patience—de toute une équipe. C'est pourquoi je tiens à citer ici tous ceux et toutes celles qui m'ont entouré de leurs précieux conseils pendant ces passionnants derniers mois.

Mes premiers remerciements ne peuvent aller qu'à Alain Lascoux, tant je lui dois dans cette "affaire". Je n'ose compter les centaines d'heures de discussion que nous avons eues ensemble, pendant lesquelles il a écrit des centaines de formules sur des centaines de feuilles pour que j'apprenne tout simplement. Je me souviens aussi des milliers de kilomètres que nous avons faits ensemble en France ou à l'Etranger, chaque séjour ayant apporté quelque chose, que ce soit une formule, un article, un programme, une collaboration ou simplement une nouvelle idée. Je ne me souviens plus des milliers de lignes de programme qu'il m'a fait réaliser mais j'espère me souvenir des milliers de choses qu'il m'a apprises. Je lui dois aussi une phrase que j'ai reprise en citation dans cette thèse pour qu'elle soit la marque du sincère respect que je lui témoigne. J'oubliais presque qu'il a accepté de présider mon jury de thèse, ce qui me fait très plaisir.

Je veux ensuite remercier Jean-Yves Thibon, qui fut successivement mon professeur en maîtrise et en DEA, mon directeur de stage de DEA et enfin mon directeur de thèse. Je me souviens de notre première discussion au sujet de la combinatoire algébrique et je dois admettre que ma première impression a été que cela devait être intéressant même si je ne comprenais rien à cet instant. Je lui dois de s'être intéressé aux programmes que j'écrivais, puis de m'avoir intéressé aux programmes qu'il souhaitait que j'écrive. Je retiendrai qu'il a été le premier à me demander de programmer des "choses" ayant pour vocation de servir à quelque chose.

Je suis tout particulièrement honoré que Meinof Geck, Michel Morvan et Piotr Pragacz aient accepté d'être les rapporteurs de cette thèse.

Je tiens à remercier Didier Arquès pour les discussions constructives que j'ai pu avoir avec lui et pour tout ce qu'il m'a apporté pour faciliter mon travail. Je le remercie chaleureusement d'avoir accepté de faire partie de mon jury.

Je suis très heureux de la présence dans mon jury de Marie-Françoise Coste-Roy et je lui suis reconnaissant de m'avoir invité à présenter "autrefois" mes travaux à Rennes.

Que Bernard Leclerc et Annick Valibouze soient également remerciés ici pour m'avoir permis de présenter mes travaux à Caen et à Paris.

C'est en quelque sorte Daniel Krob qui m'a mis sur la voie du sujet de cette thèse lorsque, pour l'examen de DEA, il proposa un choix d'articles à étudier, je me retrouvai avec les "Notes on Schubert polynomials" *! La suite, vous le comprenez, se trouve dans les pages de ce mémoire. Je lui dis donc merci.*

Je me permets encore de saluer la bonne entente des deux équipes de combinatoire à Marne-la-Vallée et à Bayreuth. Cela m'a permis d'établir de très bons contacts et des collaborations fructueuses avec Anton Betten et Alfred Wassermann.

*Je salue aussi Stéphane Bizard, Stephan Dalhem, Laurent Lheur et Brad Topol (*Georgia Institute of Technology*) qui ont largement contribué à la concrétisation de mes projets en calcul distribué.*

Merci à Vincent Prosper pour tout ce qu'il m'a apporté en général et pour le remarquable coup d'œil dont il a fait preuve en relisant ma thèse en particulier. Je lui confère volontiers le titre de "grand gourou" TEX tant j'ai apprécié son aide pour la mise en forme de ce document.

Je n'oublie surtout pas tous les membres de l'Institut Gaspard Monge avec qui nous avons festoyé grandement à plusieurs reprises. Je remercie en particulier Pierre-Yves Foucou pour toutes les réponses qu'il a apportées aux questions système que je prenais plaisir à lui poser et j'adresse des remerciements pesés à Line Fonfrede pour sa gentillesse, son efficacité et son immuable sourire face aux nombreux problèmes administratifs avec lesquels je l'ai "tourmentée".

J'aimerais aussi mettre à l'honneur l'ensemble des personnes du phalanstère de Combinatoire Algébrique—et spécialement Jacques Désarménien et Pierre-André Picon—car j'ai pris plaisir à travailler avec elles. J'en retiendrai une image très positive du travail dans la bonne humeur.

Enfin, je souhaite exprimer toute ma gratitude à Daniel Laurent, pour ce qu'il a fait pour l'Université de Marne-la-Vallée depuis son ouverture en 1989. *Inscrit dès cette année là, j'ai pu assister à la mise en place progressive des différentes structures et cela m'a permis de travailler toujours dans de bonnes conditions. Pour tout cela, je le remercie.*

Je dédie pour finir ce travail à mes proches et je leur redis toute mon affection.

Table des matières

Introduction générale

Ce n'est pas tout que commencer,
il faut savoir si la fin est bonne :
Car lors n'est pas temps d'y penser,
l'œuvre par la fin se couronne.

— ***Gilles Corrozet***, *Fables, Du Renard et du Bouc.*

La Combinatoire Algébrique est apparue officiellement en tant que rubrique 05E de la classification de 1991 de l'*American Mathematical Society*. Cette dénomination, forgée sur le même modèle que "Topologie algébrique" ou "Géométrie différentielle", désigne un courant d'idées visant à résoudre des problèmes algébriques par des méthodes combinatoires ou vice-versa. Certaines de ces idées sont très anciennes. Par exemple, l'Algèbre d'Euler, qui expose sa théorie combinatoire des partitions d'entiers, traitée au moyen de fonctions génératrices, ou la "Combinatory Analysis" de MacMahon (parue en 1915), peuvent être considérés comme des ouvrages de Combinatoire Algébrique.

L'essor actuel de la combinatoire algébrique provient d'une part de la possibilité d'expérimenter sur ordinateur et d'autre part du besoin croissant de méthodes effectives en mathématiques et dans les disciplines qui en font usage.

Il existe plusieurs logiciels permettant d'effectuer des calculs en combinatoire algébrique. Très spécifiques, ils sont souvent performants dans leur domaine. Malheureusement, ces systèmes dédiés ont le défaut de leur qualité : spécialisés dans des domaines trop précis, ils ne laissent guère de possibilités en matière d'expérimentation. Car expérimenter, c'est parfois changer de point de vue et changer de point de vue c'est souvent changer de domaine. Bien sûr, on peut aussi dire que ces logiciels ont la qualité de leur défaut puisque la performance calculatoire est l'un des objectifs visés lors du développement d'un nouveau programme. Quel est alors l'outil qui peut offrir de réelles possibilités d'expérimentation ?

Les logiciels de calcul formel se positionnent plutôt bien dans ce domaine. Ces systèmes généralistes et souples autorisent des changements de point de vue. Mais en matière de performance, toute la contradiction est là, ils ont bien du mal à rivaliser avec les programmes spécialisés.

Une solution possible consiste à développer des bibliothèques satellites de fonctions spécialisées, gravitant autour du moteur de calcul formel. Dans cet esprit, nous avons développé l'environnement `ACE`, dédié au système Maple, qui représente environ 50000 lignes de programmes. Il regroupe actuellement 16 bibliothèques permettant de manipuler les structures fondamentales de la combinatoire algébrique (permutations, mots, tableaux, fonctions symétriques) et de calculer dans les algèbres de groupe et les algèbres de Hecke des groupes symétriques et hyperoctaédraux. On y trouve ainsi les caractères et les représentations de divers groupes classiques, ainsi que des objets de création plus récente (polynômes de Schubert, fonctions symétriques non commutatives, algèbres affines quantiques, ...).

La première partie de ce mémoire est consacrée à la présentation de l'environnement ACE. Nous donnons dans le premier chapitre une rapide présentation des différentes bibliothèques. L'environnement repose sur une algorithmique dont l'objet premier est de prendre en compte des calculs algébriques via des manipulations ou des énumérations d'objets combinatoires. Il intègre une multitude de résultats et propriétés mathématiques qui lui assurent un niveau de performance respectable au sein d'un système généraliste.

Le deuxième chapitre présente des exemples combinatoires illustrant quelques possibilités de l'environnement. Nous voyons successivement quelques exemples sur les tableaux de Young et les fonctions symétriques (interprétation combinatoire des fonctions de Schur comme sommes de tableaux de Young, règle de Littlewood-Richardson). Nous donnons ensuite quelques calculs de caractères (nombre de racines k-ièmes d'une permutation, puissances symétriques et extérieures d'une représentation). Nous terminons avec le développement du q-discriminant et le calcul des fonctions génératrices des polynômes de Schubert associés au groupe symétrique et au groupe hyperoctaédral.

Le troisième chapitre présente des exemples empruntés à la géométrie. Nous montrons comment implémenter le calcul dans la cohomologie d'une grassmannienne ou d'une variété de drapeaux. Nous donnons ensuite les programmes permettant de calculer les classes de Chern et les classes de Todd de la variété de drapeaux ainsi que les degrés projectifs des variétés de Schubert. Nous voyons aussi comment calculer les polynômes de Grothendieck et les polynômes clés et nous terminons par une application de la bibliothèque FM au calcul dans l'anneau de cohomologie d'une variété grassmannienne relative.

Ce sont donc toutes les possibilités de MAPLE et toutes les capacités de l'environnement ACE qui apportent une solution au problème très précis de l'utilisation d'ordinateurs en matière d'expérimentation en combinatoire algébrique.

La deuxième partie présente les contributions mathématiques que nous avons apportées à la théorie des polynômes de Schubert. Le premier chapitre est une rapide introduction à la théorie des polynômes de Schubert. Il s'agit d'une base distinguée de l'anneau des polynômes en plusieurs variables, originellement introduite en géométrie algébrique, mais qui apparaît maintenant comme un outil combinatoire fondamental.

Nous y rappelons quelques notions sur le groupe symétrique et les opérateurs de différence divisée. Nous introduisons alors les polynômes de Schubert simples et les polynômes de Schubert doubles qui font intervenir un deuxième ensemble de paramètres indépendants.

Le deuxième chapitre présente les résultats que nous avons obtenus sur les polynômes de Schubert. Dans le cas de l'anneau des polynômes symétriques, la structure multiplicative est déterminée par la formule de Pieri qui exprime dans la base des fonctions de Schur le produit d'une fonction symétrique complète (ou élémentaire) par une fonction de Schur. Cette formule a de nombreuses interprétations en géométrie et en théorie des représentations. Sa propriété essentielle est l'absence de multiplicité.

Si l'on exprime maintenant des polynômes non nécessairement symétriques dans la base Schubert, la structure multiplicative repose sur la multiplication par une variable. Cela met en jeu une certaine combinatoire des permutations décrite par la formule de Monk.

Les polynômes de Schubert doubles forment eux-aussi une base de l'anneau des polynômes. La structure multiplicative repose à nouveau sur la multiplication des polynômes de Schubert doubles par une variable et nous donnons la formule de Monk pour les polynômes de Schubert doubles.

Lascoux et Schützenberger ont donné une "formule de Pieri" exprimant le produit d'un polynôme de Schubert simple par une fonction complète ou élémentaire. Nous donnons l'analogue de cette formule dans le cas des polynômes de Schubert doubles. Nous présentons aussi la formule de Pieri pour les polynômes de Schubert quantiques qui ont été définis plus récemment.

Nous décrivons aussi la structure multiplicative de l'anneau des polynômes vu comme module libre (de rang fini) sur l'anneau des polynômes symétriques, avec pour base les polynômes de Schubert.

La dernière partie traite de problèmes liés à la distribution de calculs. Les calculs qui nous intéressent sont souvent très demandeurs de ressources. Cela nous a conduit à orienter nos recherches sur la voie du calcul distribué. Nous présentons dans le premier chapitre plusieurs expériences que nous avons menées, notamment pour distribuer le calcul de π ou encore intégrer des routines de distribution des calculs au sein d'une bibliothèque de calcul sur entiers longs. Ces différents essais s'inscrivent dans un cadre expérimental plus vaste d'étude des possibilités de distribution de calculs dans les domaines de la combinatoire et du calcul symbolique.

Nous donnons dans le deuxième chapitre un algorithme d'énumération des tableaux de rubans de forme et de poids donnés. Les tableaux de rubans sont des généralisations des tableaux de Young et leurs énumérations sont associées au calcul de polynômes de spin généralisant les fonctions de Hall-Littlewood.

Nous donnons dans le troisième chapitre une version parallèle de cet algorithme dont nous avons effectivement tiré un programme distribuant le calcul des polynômes de spin grâce au système `PVM` qui permet de simuler une machine parallèle virtuelle par regroupement logiciel de ressources distribuées.

La fin du chapitre traite de problèmes liés à l'analyse des performances de programmes distribués sous cet environnement. Nous avons ainsi entrepris l'étude des performances de notre algorithme distribué. Puisqu'un outil acceptable faisait défaut pour mener à bien cette analyse, nous avons tout d'abord modifié un programme existant afin qu'il intègre les fonctionnalités de base que nous espérions de la part d'un traceur d'exécution. Nous avons alors utilisé notre nouvel outil afin d'analyser les performances de notre programme.

Hoc opus, hic labor est.

— ***Publius Virgilius Maro***, *L'Enéide, VI, 129.*

Première Partie

Combinatoire Algébrique et Calcul Symbolique : l'environnement ACE

Introduction

Tout l'intérêt de l'art se trouve dans le commencement.
Après le commencement, c'est déjà la fin.

— ***Pablo Picasso***, *Conversation avec E. Tériade, L'Intransigeant, 15 juin 1932.*

Le calcul formel est aujourd'hui une discipline scientifique à part entière qui s'intéresse à l'algorithmique des noyaux des systèmes de calcul. Cette discipline se distingue de ce que nous appellerons le calcul symbolique, dont l'objet est toujours la manipulation d'objets formels, mais qui se situe en termes informatiques à un niveau plus élevé. Le calcul formel fournit les outils de base fondamentaux à la mise en place de l'algorithmique mathématique.

Face aux systèmes généralistes de calcul formel, la concurrence vient de systèmes plus spécialisés. Ceux-ci ne tendent pas à proposer les mêmes services que les systèmes généralistes mais ils sont souvent nettement plus performants dans leur domaine.

Effectuer des calculs intensifs nécessite clairement une algorithmique sous-jacente optimale, le plus souvent disponible au cœur des systèmes dédiés mais que l'on trouve tout aussi bien dans des systèmes généralistes. Pourtant, la performance n'est au rendez-vous que lorsque l'algorithmique est associée à une programmation prenant en compte la nature et les spécificités des calculs. Cet investissement très lourd ne peut rentrer dans le cadre du développement de systèmes généralistes car les bibliothèques de fonctions sont des satellites et non des parties intégrantes du noyau. C'est là vraisemblablement une des différences essentielles entre les deux types de systèmes.

L'utilisation de systèmes de calcul et notamment de systèmes généralistes a apporté une dimension expérimentale nouvelle à la recherche en mathématique. L'obtention de nouveaux résultats peut désormais être liée à une démarche expérimentale qui conduit naturellement au développement de nouveaux algorithmes plus performants. Ces algorithmes sont alors intégrés au système même qui a contribué à leur élaboration.

Le logiciel MAPLE est un des logiciels disponibles sur le marché du calcul formel. Nous l'avons choisi comme support informatique pour nos calculs mais nous avons eu besoin de fonctions plus spécifiques à la combinatoire algébrique afin de pouvoir effectivement manipuler les objets mathématiques nous intéressant. Ceci nous a amené à développer l'environnement `ACE`, s'appuyant sur MAPLE et entièrement consacré aux thèmes mathématiques nous préoccupant. Globalement, ces thèmes sont directement liés à la théorie du groupe symétrique et aux objets combinatoires introduits au siècle dernier pour l'étude de ses représentations (partitions, tableaux de Young, fonctions de Schur). Ces objets apparaissent presque inévitablement dès lors que l'on aborde les aspects algorithmiques et combinatoires d'une vaste classe de structures tels que les groupes, algèbres de Lie classiques ou leurs analogues quantiques par exemple.

Avec de l'ordre de 400 fonctions dans les 16 bibliothèques spécialisées et une cinquantaine d'utilitaires dans les bibliothèques annexes, l'environnement `ACE` tente de répondre aux besoins d'expérimentation en combinatoire algébrique.

Cette partie donne un aperçu de `ACE`. Cet environnement, qui est développé depuis 1994, en est à sa deuxième version. Il trouve ses utilisateurs tant en France où il est notamment installé à Bordeaux, Caen, Paris, Rouen et Strasbourg, qu'en Allemagne, Angleterre, Australie, au Canada, en Chine, aux Etats-Unis, en Italie, en Pologne et même au Japon.

Le premier chapitre présente les objets mathématiques impliqués et quelques résultats mathématiques sous-jacents. Il brosse un portrait rapide de l'environnement en mettant en évidence les points nous paraissant intéressants.

Le deuxième chapitre montre quelques exemples combinatoires d'utilisation de `ACE`. Nous traitons tout d'abord des tableaux de Young et des fonctions symétriques (interprétation combinatoire des fonctions de Schur comme sommes de tableaux de Young, règle de Littlewood-Richardson). Nous mentionnons alors une application des fonctions symétriques au calcul de caractères (nombre de racines k-ièmes d'une permutation, puissances symétriques et extérieures d'une représentation) et terminons avec le développement du q-discriminant et le calcul des fonctions génératrices des polynômes de Schubert associés au groupe symétrique et au groupe hyperoctaédral.

Le troisième chapitre présente quelques applications de `ACE` à la géométrie énumérative. Nous expliquons comment calculer dans la cohomologie d'une grassmannienne ou d'une variété de drapeaux, puis nous donnons les programmes permettant de calculer les classes de Chern et les classes de Todd de la variété de drapeaux ainsi que les degrés projectifs des variétés de Schubert. Nous voyons ensuite comment calculer les polynômes de Grothendieck et les polynômes clés et nous terminons par des calculs dans l'anneau de cohomologie d'une variété grassmannienne relative.

Chapitre Premier

L'environnement de calcul ACE

Il ne faut pas dire que la science est utile
parce qu'elle nous permet de construire des machines.
Il faut dire : les machines sont utiles parce qu'elles
nous permettent de faire de la science.

— ***H. Poincaré.***

L'environnement `ACE` est dédié à la manipulation d'objets intervenant de manière privilégiée en combinatoire algébrique. Ce chapitre propose un tour d'horizon des différents objets mathématiques qui sont à la base de `ACE`, tour d'horizon parsemé d'exemples illustrant plutôt l'utilisation directe de MAPLE et `ACE`, que la programmation plus avancée, laquelle sera abordée dans les deux chapitres suivants. Au passage, nous nous efforcerons de mettre en évidence les points importants de la théorie sous-jacente menant à l'élaboration des algorithmes.

Nous voyons tout d'abord quelques résultats sur le groupe symétrique, puis nous abordons les différentes algèbres liées à ce groupe, *i.e.* l'algèbre du groupe, l'algèbre des différences divisées ou nilCoxeter, l'algèbre idCoxeter et enfin l'algèbre de Hecke.

Nous passons rapidement sur les bibliothèques traitant de théorie des représentations (de l'algèbre de Hecke et de l'algèbre quantique affine $U_q(\widehat{sl}_n)$), tout aussi rapidement sur la bibliothèque permettant de manipuler les caractères des groupes de Lie classiques, mais nous nous attardons plus longuement sur les fonctions symétriques et leurs analogues non commutatives.

Nous jetons ensuite un rapide coup d'œil sur les groupes hyperoctaédraux et les algèbres associées à ces groupes avant d'aborder les tableaux de Young, objets combinatoires fondamentaux.

Nous terminons sur les polynômes de Schubert qui ont trouvé une récente application dans le calcul sur l'anneau des polynômes à plusieurs variables vu comme module sur l'anneau des polynômes symétriques.

1.1 Groupes symétriques

Le groupe symétrique (`SG`) $\mathfrak{S}_n$ est le groupe des $n!$ permutations (`ListPerm`) de l'ensemble $[1, n] := \{1, 2, \ldots, n\}$. Défini pour tout $n > 0$, le groupe $\mathfrak{S}_n$ est plongé dans $\mathfrak{S}_m$ pour tout $m > n$ puisque l'on peut identifier $\mathfrak{S}_n$ au sous-groupe de $\mathfrak{S}_m$ en ajoutant à toute permutation $w \in \mathfrak{S}_n$ les points fixes $n+1, n+2, \ldots, m$. Grâce à ce plongement $\mathfrak{S}_n \hookrightarrow \mathfrak{S}_m$, on peut définir le groupe $\mathfrak{S}_\infty$ de toutes les permutations de l'ensemble des entiers positifs qui fixent tous ces entiers sauf un nombre fini d'entre eux :

$$\mathfrak{S}_\infty := \bigcup_{i>0} \mathfrak{S}_i \quad .$$

Nous noterons une permutation $w \in \mathfrak{S}_n$ par la liste des images de $\{1, 2, \ldots, n\}$ par w, alors vue comme une bijection de $[1, n]$ sur lui-même : $w := [w(1), w(2), \ldots, w(n)]$.

Une statistique élémentaire sur les permutations est le *nombre d'inversions* : étant donnée une permutation $w \in \mathfrak{S}_n$, une *inversion* est un couple de positions (i, j) tel que $1 \leq i < j \leq n$ et $w(i) > w(j)$. Une *descente* est une inversion sur deux positions consécutives (`Perm2Desc` produit l'ensemble des descentes d'une permutation) de la permutation et inversement, une *montée* (la fonction `Perm2Rise` donne l'ensemble des montées d'une permutation donnée) de w est une position i telle que $w(i) < w(i+1)$.

La *longueur* $l(w)$ (`Perm2Length`) de w est le nombre de ses inversions, *i.e.* le cardinal de l'ensemble $I(w)$ (la fonction `Inversions` calcule cet ensemble) défini par :

$$I(w) := \{(i, j), 1 \leq i < j \leq n \text{ et } w(i) > w(j)\} \quad .$$

L'élément $[n, n-1, \ldots, 1]$ de $\mathfrak{S}_n$ est appelé la *permutation maximale* (`LastPerm`) de $\mathfrak{S}_n$ car il comporte le maximum d'inversions pour le groupe $\mathfrak{S}_n$, c'est-à-dire le binomial $\binom{n}{2}$.

Cette notion d'inversion est très importante puisqu'elle conduit à un autre codage des éléments du groupe symétrique, ce codage facilitant parfois la description d'opérateurs sur le groupe. Etant donnée une permutation $w \in \mathfrak{S}_n$, le *code* $c(w)$ (`Perm2Code`) de w est un vecteur d'entiers positifs ou nuls tel que sa i-ième composante est le nombre de positions $j > i$ telles que $w(j) < w(i)$.

Par exemple le code de la permutation $w := [2, 6, 4, 3, 1, 7, 5]$ est $c(w) := [1, 4, 2, 1, 0, 1, 0]$ et l'exemple suivant donne les codes de toutes les permutations de $\mathfrak{S}_4$:

```
ACE> map(Perm2Code, ListPerm(4));

      [[0, 0, 0, 0], [0, 2, 0, 0], [0, 0, 1, 0], [3, 0, 0, 0], [0, 1, 0, 0],

       [0, 2, 1, 0], [0, 1, 1, 0], [3, 0, 1, 0], [2, 0, 0, 0], [2, 2, 0, 0],

       [2, 0, 1, 0], [3, 2, 0, 0], [1, 0, 0, 0], [1, 2, 0, 0], [1, 0, 1, 0],

       [3, 1, 0, 0], [1, 1, 0, 0], [1, 2, 1, 0], [1, 1, 1, 0], [3, 1, 1, 0],

       [2, 1, 0, 0], [2, 2, 1, 0], [2, 1, 1, 0], [3, 2, 1, 0]]
```

Une permutation $w \in \mathfrak{S}_n$ peut être représentée par une matrice $M(w)$ carrée d'ordre n de 0 et de 1 (`Perm2Mat`), les entrées non nulles étant les $M[r, w(r)] = 1$, $r = 1, \ldots, n$. A chaque 1 de la matrice, on associe l'équerre formée des entrées à droite du 1 (sur la même ligne) et en dessous du 1 (sur la même colonne). Le diagramme $D(w)$ de w est le complémentaire dans $[1,n]\times[1,n]$ de l'union des équerres. Nous représenterons chacun des points de $D(w)$ par un "$\square$" et chacun des points des équerres par un "$\blacksquare$". L'exemple suivant illustre la construction du diagramme $D(w)$ (à droite sur la figure) à partir de la matrice $M(w)$ (à gauche) pour la permutation $w := [2,6,4,3,1,7,5]$:

$$\begin{bmatrix} 0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&1&0&0&0\\ 0&0&1&0&0&0&0\\ 1&0&0&0&0&0&0\\ 0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0 \end{bmatrix} \quad \begin{bmatrix} 0&1&\blacksquare&\blacksquare&\blacksquare&\blacksquare&\blacksquare\\ 0&\blacksquare&0&0&0&1&\blacksquare\\ 0&\blacksquare&0&1&\blacksquare&\blacksquare&\blacksquare\\ 0&\blacksquare&1&\blacksquare&\blacksquare&\blacksquare&\blacksquare\\ 1&\blacksquare&\blacksquare&\blacksquare&\blacksquare&\blacksquare&\blacksquare\\ \blacksquare&\blacksquare&\blacksquare&\blacksquare&0&\blacksquare&1\\ \blacksquare&\blacksquare&\blacksquare&\blacksquare&1&\blacksquare&\blacksquare \end{bmatrix} \quad \begin{bmatrix} \square&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\ \square&\cdot&\square&\square&\square&\cdot&\cdot\\ \square&\cdot&\square&\cdot&\cdot&\cdot&\cdot\\ \square&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot&\square&\cdot&\cdot\\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \end{bmatrix}$$

Remarque. Le code d'une permutation w peut aussi être défini comme la suite des nombres de points du diagramme dans ses lignes successives. Dans l'exemple précédent, le code est $[1,4,2,1,0,1,0]$. Cela montre notamment le lien étroit existant entre les différents codages, chacun d'eux ayant son intérêt propre pour la description d'algorithmes.

Notons $s_i, i = 1,\ldots,n-1$ la transposition élémentaire échangeant i et $i+1$. Les transpositions élémentaires satisfont les relations (dites de tresse) suivantes :

$$s_i s_j = s_j s_i \qquad \text{si } |i-j|>1 \quad , \tag{1.1}$$

$$s_i s_{i+1} s_i = s_{i+1} s_i s_{i+1} \qquad 1 \le i \le n-2 \quad , \tag{1.2}$$

ainsi que les relations supplémentaires :

$$s_i^2 = 1 \quad . \tag{1.3}$$

Plus généralement, on peut remplacer (1.3) par les relations quadratiques :

$$(s_i - q_1)(s_i - q_2) = 0 \quad . \tag{1.4}$$

L'algèbre engendrée par les relations de tresse et (1.4) est appelée algèbre de Hecke (générique) du groupe symétrique (bibliothèque `HEKA`). Cette notion est définie pour tout groupe de Coxeter [27]. Par spécialisation de q_1 et q_2, on obtient d'autres algèbres qui font l'objet de bibliothèques de fonctions particulières : la bibliothèque `NCA` permet un calcul dans l'algèbre nilCoxeter et la bibliothèque `IDCA` dans l'algèbre idCoxeter.

On utilise très souvent la décomposition d'une permutation en produit de transpositions élémentaires. Un produit de longueur minimale est appelé une *décomposition réduite*. La fonction `Perm2Rd` prend en argument une permutation et rend une décomposition réduite canonique, mais cette décomposition n'est pas unique en général et la fonction `Perm2ListRd` produit quant à elle la liste de toutes les décompositions réduites d'une permutation donnée. L'exemple suivant illustre l'utilisation de ces deux fonctions :

```
ACE> Perm2Rd([6,5,4,3,2,1]);

                  [5, 4, 3, 2, 1, 5, 4, 3, 2, 5, 4, 3, 5, 4, 5]

ACE> Perm2ListRd([3,1,5,4,2]);

     [[2, 3, 4, 3, 1], [4, 2, 3, 4, 1], [2, 4, 3, 4, 1], [2, 3, 4, 1, 3],

      [2, 3, 1, 4, 3], [2, 1, 3, 4, 3], [4, 2, 3, 1, 4], [2, 4, 3, 1, 4],

      [4, 2, 1, 3, 4], [2, 4, 1, 3, 4], [2, 1, 4, 3, 4]]
```

Le nombre de décompositions réduites est donné par la fonction **Perm2NbRd** comme le montre l'exemple suivant :

```
ACE> Perm2NbRd([6,5,4,3,2,1]);

                                  292864
```

Nous avons adopté les conventions suivantes pour multiplier les permutations. Etant donné w et une transposition élémentaire s_i :

$$w\,s_i := [w(1), \ldots, w(i+1), w(i), \ldots, w(n)] \quad .$$

Remarque. Les conventions ne sont pas uniformes dans la littérature, ce qui est source de confusion. Pour nous, la multiplication par s_i à droite permute les entrées de w aux places i et $i+1$, alors que la multiplication par s_i à gauche échange les valeurs i et $i+1$.

Le parcours du diagramme $D(w)$ fournit aisément une décomposition réduite de w. On remplace dans chaque ligne les points successifs du diagramme par des entiers consécutifs, en commençant par l'indice de la ligne. Lisant ce diagramme de droite à gauche et de haut en bas, et interprétant i comme s_i, on obtient une décomposition réduite de w.

Ajoutant à ce diagramme les entrées non nulles de $M(w)$ figurées par des "x", on obtient le *diagramme de Rothe* de w. La fonction **Perm2Rothe** permet de calculer ce diagramme comme le montre l'exemple suivant pour la permutation $w = [2, 6, 4, 3, 1, 7, 5]$:

```
ACE> Perm2Rothe([2,6,4,3,1,7,5]);

                        [ 1  x  .  .  .  .  . ]
                        [                     ]
                        [ 2  .  3  4  5  x  . ]
                        [                     ]
                        [ 3  .  4  x  .  .  . ]
                        [                     ]
                        [ 4  .  x  .  .  .  . ]
```

```
[                     ]
[ x  .  .  .  .  .  . ]
[                     ]
[ .  .  .  .  6  .  x ]
[                     ]
[ .  .  .  .  x  .  . ]
```

diagramme à partir duquel on déduit la décomposition réduite de w :

$$w = s_1\ s_5 s_4 s_3 s_2\ s_4 s_3\ s_4\ s_6 \quad .$$

L'ensemble des permutations ordonnées par longueur, muni des arêtes (w, ws_i) est appelé le *permutoèdre* et l'ordre induit est dit *ordre faible*. La figure 1.1 page 15 montre le permutoèdre de $\mathfrak{S}_4$. On y voit bien notamment les relations de commutation de type (1.1), *i.e.* pour $\mathfrak{S}_4$ uniquement la relation $s_1 s_3 = s_3 s_1$ ainsi que les deux relations de tresse $s_1 s_2 s_1 = s_2 s_1 s_2$ et $s_2 s_3 s_2 = s_3 s_2 s_3$.

Si l'on ajoute au permutoèdre des arêtes entre les permutations de longueurs consécutives qui diffèrent par une transposition (*i.e.* échangeant deux valeurs), on construit alors le *Bruhatoèdre*. L'ordre associé est appelé *ordre fort* ou *ordre de Bruhat*. L'intervalle des permutations plus petites qu'une permutation donnée (pour l'ordre de Bruhat) est obtenu grâce à la fonction `Interval` qui renvoie les codes des permutations correspondantes :

```
ACE> Interval([3,4,2,1]);

     [[2, 2, 1, 0], [2, 2, 0, 0], [2, 1, 1, 0], [2, 1, 0, 0], [2, 0, 1, 0],

      [2, 0, 0, 0], [1, 2, 1, 0], [1, 2, 0, 0], [1, 1, 1, 0], [1, 1, 0, 0],

      [1, 0, 1, 0], [1, 0, 0, 0], [0, 2, 1, 0], [0, 2, 0, 0], [0, 1, 1, 0],

      [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0]]
```

codes à partir desquels on peut sans peine, si l'on préfère, retrouver les permutations comme le montre la commande suivante :

```
ACE> map(Code2Perm, ");

     [[3, 4, 2, 1], [3, 4, 1, 2], [3, 2, 4, 1], [3, 2, 1, 4], [3, 1, 4, 2],

      [3, 1, 2, 4], [2, 4, 3, 1], [2, 4, 1, 3], [2, 3, 4, 1], [2, 3, 1, 4],

      [2, 1, 4, 3], [2, 1, 3, 4], [1, 4, 3, 2], [1, 4, 2, 3], [1, 3, 4, 2],

      [1, 3, 2, 4], [1, 2, 4, 3], [1, 2, 3, 4]]
```

Par ailleurs la fonction `Betti` retourne la fonction génératrice du nombre de permutations dans un intervalle donné en fonction de leurs longueurs :

```
ACE> Betti([3,4,2,1], 'q');

                              5      4      3      2
                             q  + 3 q  + 5 q  + 5 q  + 3 q + 1
```

cet exemple nous précisant notamment qu'il y a 3 permutations de longueur 4 dans l'intervalle de $[3, 4, 2, 1]$.

Remarque. Ce polynôme correspond en termes géométriques au polynôme de Poincaré :

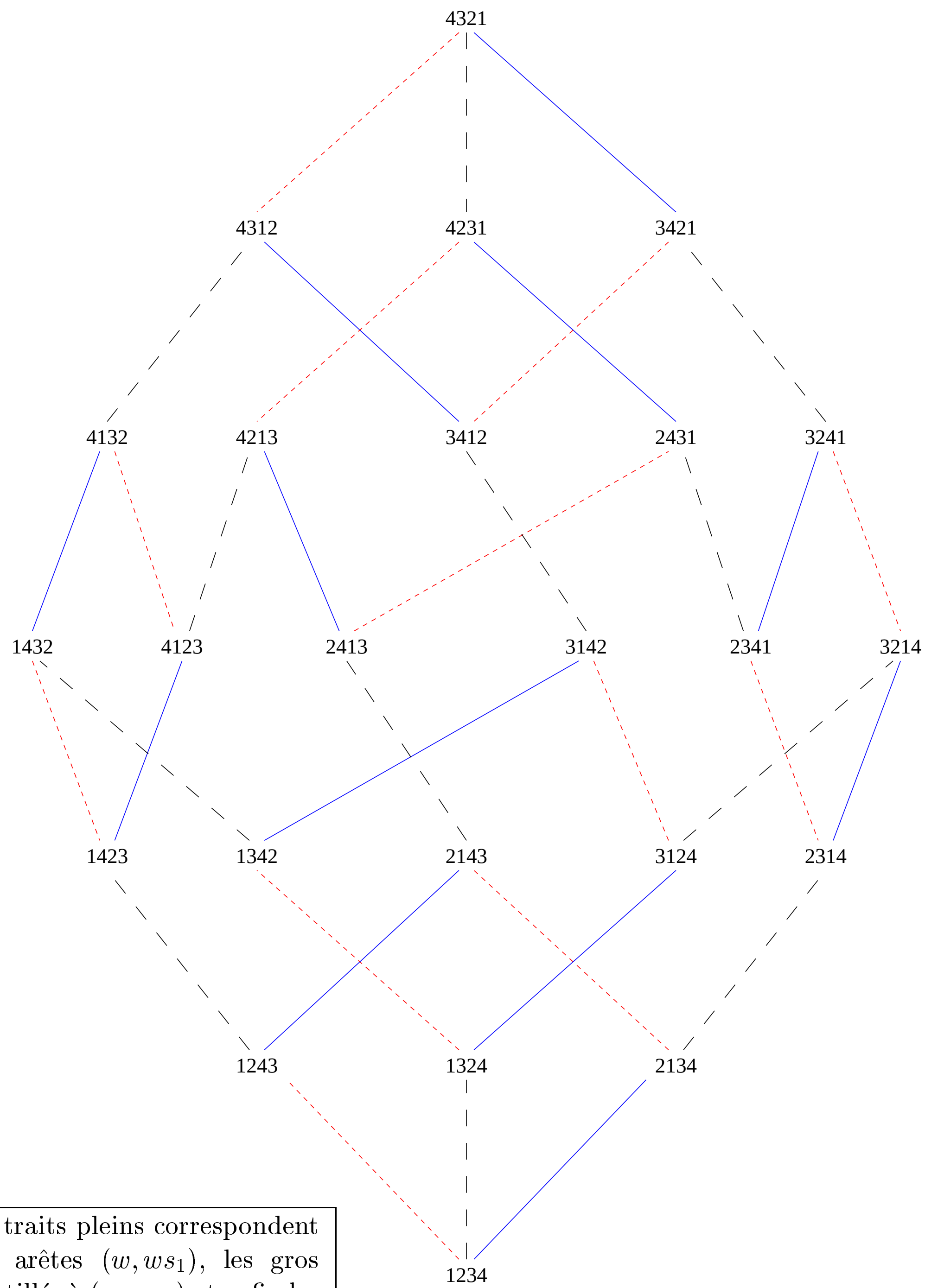
$$\sum q^k \dim H^{2k}(\overline{\Omega}_{3,4,2,1}, \mathbb{Q}) \quad ,$$

dont les coefficients sont les nombres de Betti de la variété de Schubert $\overline{\Omega}_{3,4,2,1}$ de la variété $\mathcal{F}_4$ des drapeaux complets de $\mathbb{C}^4$.

La bibliothèque `SG` propose les outils pour engendrer les permutations (que ce soit en ordre lexicographique, lexicographique inverse ou par longueur dans le permutoèdre, ou seulement certaines permutations dites dominantes, grassmanniennes ou vexillaires), comparer des permutations (que ce soit pour l'ordre faible ou l'ordre de Bruhat), transformer des permutations (de la notation usuelle au code et inversement, de la permutation au diagramme de Rothe ou vice-versa, etc.) et calculer sur les permutations quelques statistiques utiles telles que les descentes, les montées, la signature d'une permutation ou le nombre de décompositions réduites. Expérimenter, c'est par exemple chercher les lois qui régissent les nombres de décompositions réduites des permutations maximales des différents groupes symétriques, c'est-à-dire donner le nombre de chemins dans le permutoèdre partant du sommet et descendant (pour le nombre d'inversions) vers la permutation identité. L'exemple suivant affiche la suite de ces nombres pour les groupes symétriques $\mathfrak{S}_1$ à $\mathfrak{S}_9$:

```
> seq(Perm2NbRd(LastPerm(i)) , i=1..9);

    1, 1, 2, 16, 768, 292864, 1100742656, 48608795688960, 29258366996258488320
```

En fait, cette formule est connue non seulement dans le cas particulier de la permutation maximale mais aussi pour une classe plus vaste de permutations (les permutations vexillaires). Pour ces dernières, c'est la formule des équerres (7.2), que nous abordons dans un autre contexte au chapitre 7, qui donne précisément le nombre de décompositions réduites. Ceci est loin d'épuiser le sujet car la probabilité qu'une permutation soit vexillaire tend vers 0 lorsque n croît !



Les traits pleins correspondent aux arêtes (w, ws_1), les gros pointillés à (w, ws_2) et enfin les petits pointillés à (w, ws_3).

Figure 1.1: *Permutoèdre du groupe symétrique* $\mathfrak{S}_4$.

1.2 Algèbre du groupe symétrique

L'algèbre du groupe symétrique $\mathbb{K}[\mathfrak{S}_n]$ (bibliothèque `SGA`) où $\mathbb{K}$ désigne un anneau de coefficients est définie comme l'algèbre engendrée par les transpositions élémentaires $s_i, i = 1, \ldots, n-1$. De fait, une base linéaire naturelle de cette algèbre consiste en les permutations de $\mathfrak{S}_n$, et c'est cette base que nous avons adoptée pour représenter dans `ACE` les éléments de l'algèbre. Ces éléments sont ainsi des sommes formelles de produits d'une permutation et d'un élément de l'anneau des coefficients.

La bibliothèque donne certains éléments distingués de l'algèbre du groupe symétrique, comme par exemple les éléments de *Jucis-Murphy* définis par :

$$\texttt{Jucis}(k) := \sum_{i \leq k-1} t_{ik} \quad ,$$

la somme étant prise sur toutes les transpositions échangeant i et k pour $i = 1, \ldots, k-1$:

```
ACE> Jucis(4, 6);

        [4, 2, 3, 1, 5, 6] + [1, 4, 3, 2, 5, 6] + [1, 2, 4, 3, 5, 6]
```

L'algèbre engendrée par les éléments de Jucis-Murphy est une sous-algèbre commutative maximale de l'algèbre du groupe symétrique, les fonctions symétriques en ces éléments engendrant le centre de l'algèbre de groupe $\mathbb{K}[\mathfrak{S}_n]$. Cette algèbre contient une sous-famille $\{e_t\}$ d'idempotents mutuellement orthogonaux, où t parcourt l'ensemble des tableaux standards de poids n. La correspondance $t \longrightarrow e_t$ s'obtient grâce à la fonction `StdTab2Idemp` :

```
ACE> StdTab2Idemp([[3], [1,2,4]]);

 - 1/48 [1, 4, 2, 3] + 1/24 [1, 2, 4, 3] + 5/48 [1, 4, 3, 2] - 1/48 [1, 3, 4, 2]

 - 1/12 [3, 4, 1, 2] - 1/48 [3, 1, 4, 2] - 1/48 [2, 4, 1, 3] + 1/24 [2, 1, 4, 3]

 + 5/48 [2, 4, 3, 1] - 1/48 [2, 3, 4, 1] - 1/12 [3, 4, 2, 1] - 1/48 [3, 2, 4, 1]

 + 1/8 [1, 2, 3, 4]  - 1/48 [4, 1, 2, 3] - 1/16 [1, 3, 2, 4] + 5/48 [4, 1, 3, 2]

 - 1/16 [3, 1, 2, 4] - 1/12 [4, 3, 1, 2] + 1/8 [2, 1, 3, 4]  - 1/48 [4, 2, 1, 3]

 - 1/16 [2, 3, 1, 4] + 5/48 [4, 2, 3, 1] - 1/16 [3, 2, 1, 4] - 1/12 [4, 3, 2, 1]
```

Les deux fonctions `Carre` et `Nabla` donnent d'autres bases linéaires distinguées de $\mathbb{K}[\mathfrak{S}_n]$. On peut d'ailleurs construire des idempotents à partir de ces éléments : par exemple, `Carre(LastPerm(`n`))` est la somme de toutes les permutations de $\mathfrak{S}_n$ et `Nabla(LastPerm(`n`))` est la somme alternée de tous les éléments de $\mathfrak{S}_n$. Plus généralement, étant donnée une forme, par exemple $(3,2)$, on obtiendra un idempotent, à un scalaire près, en effectuant dans l'algèbre du groupe symétrique le produit[1] `Carre`$([2,4,1,3,5]) \times$ `Nabla`$([2,5,1,4,3])$ que l'on déduit de la lecture ligne des tableau et contre-tableau spéciaux suivants :

2	4	
1	3	5

2	5
1	4
	3

La bibliothèque permet aussi de réaliser cette algèbre du groupe symétrique comme une algèbre d'opérateurs sur l'anneau des polynômes (la fonction `SgaOnPol` donne l'action d'un élément de l'algèbre sur un polynôme, l'action d'une permutation consistant à permuter les variables du polynôme).

Nous verrons par la suite comment considérer l'anneau des polynômes comme un module libre sur l'anneau des polynômes symétriques. Ce point de vue trouve notamment un intérêt dans le contexte de l'action de l'algèbre du groupe symétrique sur l'anneau des polynômes. En effet, les polynômes symétriques sont par définition invariants par permutation des variables et c'est pourquoi la bibliothèque `SGA` dispose en plus de la fonction `SgaOnXfix` pour tenir compte de cette invariance. Nous reviendrons plus longuement sur cette structure lorsque nous parlerons de la bibliothèque `FM`.

1.3 Algèbre nilCoxeter

La bibliothèque `NCA` offre quelques outils de calcul dans l'algèbre nilCoxeter ou algèbre des différences divisées. Cette algèbre est réalisée comme l'algèbre des différences divisées agissant sur l'anneau des polynômes. Soit f une fonction en les variables $\ldots, x_i, x_{i+1}, \ldots$, la différence divisée de f par rapport aux variables x_i et x_{i+1} est définie depuis Newton par :

$$\partial_{x_i,x_{i+1}}(f) := \frac{f(\ldots,x_i,x_{i+1},\ldots) - f(\ldots,x_{i+1},x_i,\ldots)}{x_i - x_{i+1}} \quad . \tag{1.5}$$

Cet opérateur envoie un polynôme de degré d sur un polynôme de degré $d-1$, symétrique en x_i et x_{i+1}[2].

On se place désormais dans l'anneau $\mathbb{Z}[x_1, x_2, \ldots]$ des polynômes en une infinité de variables. On pose alors pour chaque $i \geq 1$:

$$\partial_i \quad := \quad \partial_{x_i,x_{i+1}} \quad . \tag{1.6}$$

En 1973, Bernstein, Gelfand & Gelfand [28] et Demazure [10] ont établi que ces opérateurs satisfont aux relations de tresse :

$$\begin{aligned} \partial_i\partial_j &= \partial_j\partial_i & \text{si } |i-j| > 1 \quad , \qquad (1.7)\\ \partial_i\partial_{i+1}\partial_i &= \partial_{i+1}\partial_i\partial_{i+1} & 1 \leq i \leq n-2 \quad , \qquad (1.8)\end{aligned}$$

l'équation supplémentaire (1.3) étant remplacée par :

$$\partial_i^2 \quad = \quad 0 \quad . \tag{1.9}$$

(1) Ce produit étant réalisé par l'opérateur MAPLE `&!*`.

(2) Le polynôme $f(\ldots,x_i,x_{i+1},\ldots) - f(\ldots,x_{i+1},x_i,\ldots)$ étant par construction anti-symétrique, il contient bien le facteur $(x_i - x_{i+1})$.

De fait, les relations (1.7) et (1.8) entraînent que l'on peut associer de manière non ambiguë à chaque permutation $w \in \mathfrak{S}_n$ une différence divisée ∂_w : si une décomposition réduite de w est $w := s_{i_1} s_{i_2} \dots s_{i_k}$, on pose :

$$\partial_w := \partial_{i_1} \partial_{i_2} \cdots \partial_{i_k} \quad . \tag{1.10}$$

✍**Remarque**. Le cas de Newton ∂_i correspond au cas d'une transposition élémentaire s_i et peut être noté ∂_{s_i}.

L'algèbre engendrée par les ∂_i est appelée l'algèbre nilCoxeter ou algèbre des différences divisées. Une base linéaire de cette algèbre est constituée des ∂_w, w étant une permutation. Deux types de calcul sont disponibles via la bibliothèque `NCA` :

① un calcul formel impliquant uniquement les relations (1.7), (1.8) et (1.9) ;

② un calcul par action concrète sur l'anneau des polynômes avec notamment la fonction `NcaOnPol`.

La bibliothèque permet également de calculer une autre base de l'algèbre. Cette base appelée base de *Yang-Baxter* [47] est obtenue grâce à la fonction `NcaYang` en remplaçant les relations de tresse (1.8) par les équations de *Yang-Baxter* :

$$Y_i(u) Y_{i+1}(u+v) Y_i(v) = Y_{i+1}(v) Y_i(u+v) Y_{i+1}(u) \quad ,$$

qui impliquent des paramètres supplémentaires (n paramètres dans le cas de $\mathfrak{S}_n$).

✍**Remarque**. Soit un polynôme $f \in \mathbb{K}[x_1]$ de degré inférieur à n en l'unique variable x_1 et $\{y_1, \dots, y_{n-1}\}$ un ensemble de paramètres dans $\mathbb{K}$. La formule d'interpolation de Newton peut être écrite de la manière suivante :

$$f(x_1) = f(y_1) + \sum_{i=1}^{n-1} (x_1 - y_1) \cdots (x_1 - y_i) \; \partial_i \cdots \partial_1(f) \quad , \tag{1.11}$$

dans laquelle les différences divisées n'agissent que sur l'alphabet $\{y_1, \dots, y_{n-1}\}$. De fait, les différences divisées ont été introduites en mathématiques pour interpoler les fonctions d'une variable. La bibliothèque `SP` fournit une généralisation de la formule de Newton dans le cas de plusieurs variables.

1.4 Algèbre idCoxeter

Comme dans le cas nilCoxeter, on considère une fonction f des variables $\dots, x_i, x_{i+1}, \dots$ et l'on définit la différence divisée isobare de f par rapport à x_i et x_{i+1} par :

$$\pi_{x_i, x_{i+1}}(f) := \frac{f(\dots, x_i, x_{i+1}, \dots) - f(\dots, x_{i+1}, x_i, \dots)}{x_i / x_{i+1} - 1} \quad . \tag{1.12}$$

Cet opérateur qui à nouveau envoie un polynôme sur un polynôme préserve cette fois le degré. Comme précédemment, on définit pour chaque $i \geq 1$:

$$\pi_i \quad := \quad \pi_{x_i, x_{i+1}} \quad , \tag{1.13}$$

et cet opérateur satisfait à nouveau les relations de tresse, la relation quadratique supplémentaire étant désormais :

$$\pi_i^2 \quad = \quad -\pi_i \quad . \tag{1.14}$$

Par suite, on peut définir des opérateurs plus généraux π_w indicés par les permutations w, par produit d'opérateurs simples. Ces opérateurs π_w forment alors une base de l'algèbre engendrée par les π_i, appelée algèbre idCoxeter ou 0-algèbre de Hecke.

La bibliothèque `IDCA` propose les fonctions de calcul formel dans cette algèbre (le produit étant donné par l'opérateur `&$*`) mais réalise aussi l'algèbre idCoxeter concrètement comme une algèbre d'opérateurs agissant sur l'anneau des polynômes. Par exemple, la fonction `IdcaOnPol` prend en premier argument un élément de l'algèbre qu'elle fait agir sur son deuxième argument de type polynôme :

```
ACE> IdcaOnPol(q^2*[3,1,2], x1^2*x2*y1^2*y2^3);

                                 2            3   2
                                q  (x2 + x3) y2  y1  x1 x3
```

On dispose aussi pour cette algèbre d'une autre base distinguée, liée à l'équation de Yang-Baxter, que la fonction `IdcaYang` permet de calculer :

```
ACE> IdcaYang([3,2,1]);

        (x1 - x3) [2, 1, 3]   (x1 - x2) (x2 - x3) (x1 - x3) [3, 2, 1]
        ------------------- + ---------------------------------------
                x1                               2
                                               x1  x2

           (x1 - x3) [1, 3, 2]   (x1 - x3) (x1 - x2) [2, 3, 1]
         + ------------------- + -----------------------------
                   x1                          2
                                             x1

           (x1 - x3) (x2 - x3) [3, 1, 2]
         + ----------------------------- + [1, 2, 3]
                      x1 x2
```

1.5 Algèbre de Hecke

L'algèbre de Hecke générique est une déformation de l'algèbre du groupe symétrique qui implique deux paramètres q_1 et q_2. Les opérateurs simples τ_i agissent maintenant de la manière suivante :

$$\tau_i(f) := (q_1 x_2 + q_2 x_1)\,\partial_i(f) + q_1\, f \quad . \tag{1.15}$$

Ils satisfont les relations de tresse et le carré d'un opérateur est égal à :

$$\tau_i^2 \quad = \quad \tau_i\,(q_1 + q_2) - q_1 q_2 \quad . \tag{1.16}$$

On retrouve l'algèbre de Hecke usuelle en posant $q_1 = q$ et $q_2 = -1$. L'algèbre nilCoxeter est obtenue pour $q_1 = q_2 = 0$ et l'algèbre idCoxeter pour $q_1 = 0$, $q_2 = -1$.

Par produits d'opérateurs élémentaires, on obtient les éléments τ_w indicés par des permutations w. Ces derniers constituent une base linéaire de l'algèbre de Hecke générique qui est définie comme l'algèbre engendrée par les τ_i.

La bibliothèque `HEKA` propose les fonctions basiques de calcul dans l'algèbre et inclut par ailleurs, comme dans le cas des algèbres du groupe symétrique, nilCoxeter et idCoxeter une fonction—`HekaYang`—permettant de calculer la base de Yang-Baxter de l'algèbre alors vue comme module libre à coefficients en les variables x_i :

```
ACE> HekaYang([3,1,2]);

        (- x3 + x2) [1, 3, 2]                  (x3 - x1) [2, 1, 3]
      - --------------------- + [1, 2, 3] + -------------------
             x2 (q1 + q2)                         x1 (q1 + q2)

                   2
          (- x3  + x3 x1 + x3 x2 - x1 x2) [3, 1, 2]
        - ----------------------------------------
                                      2
                       x1 (q1 + q2)  x2
```

Comme c'était déjà le cas pour les algèbres précédentes, l'action d'un élément arbitraire de l'algèbre sur un polynôme est prise en compte par la fonction `HekaOnPol` :

```
ACE> HekaOnPol([2,1], x1);

                         x2 q1 + x1 q2 + q1 x1
```

1.6 Représentations de l'algèbre de Hecke et isomorphismes

La bibliothèque `SHRI`, développée par Eric Laugerotte, de l'Université de Rouen, fournit des fonctions relatives aux représentations irréductibles des algèbres de Hecke.

On peut construire une famille complète de représentations de l'algèbre de Hecke comme orbites d'une q-déformation du déterminant de Vandermonde. Pour un vecteur d'entiers $I \in \mathbb{N}^r$, on pose $j_0 = 0$, $j_1 = i_1$, $j_2 = i_1 + i_2$, …, $j_r = i_1 + \cdots + i_r$. Le *q-Vandermonde* d'indice I est alors par définition :

$$\Delta_I := \prod_{m=0}^{r-1} \prod_{j_m < k < \ell \leq j_{m+1}} (q x_\ell - x_k) \quad .$$

La projection sur l'espace des monômes dont l'exposant est un mot de Yamanouchi de l'espace engendré par l'action sur Δ_I de l'algèbre de Hecke définie au paragraphe précédent, est une représentation irréductible dite q-module de Specht.

La bibliothèque `SHRI` permet aussi de choisir entre les différentes paramétrisations (q_1, q_2), $(q, -1)$, $(v, -\frac{1}{v})$. Par exemple, les coefficients de la table des caractères de l'algèbre de Hecke générique peut être calculée par la fonction `ShriCharTable` grâce à une q-version de la règle de Murnaghan-Nakayama, due à Désarménien. L'exemple suivant illustre ce calcul pour $\mathfrak{S}_4$:

```
ACE> ShriCharTable(4);

              [     3          2                 2                        ]
              [   q2         q2                q2               q2     1 ]
              [                                                           ]
              [      2                                                    ]
              [ q1 q2   q2 (q1 + q2)  q2 q1 + q2 (q1 + q2)  2 q2 + q1  3 ]
              [                                                           ]
              [                                2     2                    ]
              [    0        q2 q1            q2   + q1        q1 + q2  2 ]
              [                                                           ]
              [   2                                                       ]
              [ q1   q2  q1 (q1 + q2)  q1 (q1 + q2) + q2 q1  2 q1 + q2  3 ]
              [                                                           ]
              [     3          2                 2                        ]
              [   q1         q1                q1               q1     1 ]
```

La bibliothèque fournit aussi des q-déformations des éléments de Jucis-Murphy qui engendrent une sous-algèbre commutative maximale contenant un système complet d'idempotents orthogonaux fournis par la fonction `ShriIdemp`.

1.7 Représentation de l'algèbre $\mathrm{U_q}(\widehat{\mathrm{sl}}_\mathrm{n})$ dans l'espace de Fock

La bibliothèque `FOCK` a été développée par Bernard Leclerc (Université de Caen) et Jean-Yves Thibon (Université de Marne-la-Vallée). Elle permet de calculer avec les représentations de l'algèbre affine quantique $U_q(\widehat{sl}_n)$ dans l'espace de Fock.

Partant de [63], on identifie l'espace de Fock avec l'espace vectoriel $\mathcal{F}$ engendré par les diagrammes de Young λ. En fait, il peut être intéressant de regarder $\mathcal{F}$ comme l'espace des fonctions symétriques en identifiant λ à la fonction de Schur s_λ. Les principales routines de cette bibliothèque permettent de calculer :

① l'action des générateurs e_i, f_i de $U_q(\widehat{sl}_n)$ sur une combinaison linéaire de diagrammes de Young ;

② le graphe cristallin de $\mathcal{F}$ et du $U_q(\widehat{sl}_n)$-module $M(\Lambda_0)$ plongé dans $\mathcal{F}$;

③ les bases cristallines globales de $M(\Lambda_0)$ et de $\mathcal{F}$.

Il a été conjecturé dans [46] et démontré ensuite par Ariki [1] que les matrices exprimant la base canonique globale $G(\mu)$ (pour μ parcourant les partitions n-régulières de poids m) sur la base des fonctions de Schur, étaient des q-analogues de la matrice de décomposition de l'algèbre de Hecke $\mathcal{H}_m(\zeta)$, pour ζ une racine primitive n-ième de l'unité. Cette théorie a ensuite été étendue aux q-algèbres de Schur. Par exemple, la matrice de décomposition de la (-1)-algèbre de Schur $\mathcal{S}_{-1}(5)$ est obtenue par :

```
ACE> subs(q=1, CBMat(2, 5));

                         [ 1  0  0  0  0  0  0 ]
                         [                     ]
                         [ 0  1  0  0  0  0  0 ]
                         [                     ]
                         [ 0  0  1  0  0  0  0 ]
                         [                     ]
                         [ 1  0  1  1  0  0  0 ]
                         [                     ]
                         [ 0  0  1  1  1  0  0 ]
                         [                     ]
                         [ 0  1  0  0  0  1  0 ]
                         [                     ]
                         [ 1  0  0  1  0  0  1 ]
```

1.8 Caractères des groupes de Lie classiques

La bibliothèque `CG` a été écrite par Renaud Eppstein, de l'Université de Marne-la-Vallée. Elle propose un outil de calcul en théorie des représentations des groupes symplectiques et orthogonaux. Elle permet de décomposer des produits tensoriels, de calculer les dimensions des représentations et les règles de branchement $GL(n) \downarrow O(n)$ et $GL(n) \downarrow Sp(n)$. Les algorithmes sous-jacents reposent sur l'emploi des fonctions de Schur symplectiques et orthogonales que l'on peut aussi exprimer sur les autres bases usuelles des fonctions symétriques. Par exemple :

$$Sp_\mu = \begin{vmatrix} h_{\mu_1} & h_{\mu_1+1} + h_{\mu_1-1} & \dots & h_{\mu_1-1+n} + h_{\mu_1+1-n} \\ h_{\mu_2-1} & h_{\mu_2} + h_{\mu_2-2} & \dots & h_{\mu_2-2+n} + h_{\mu_2-n} \\ \vdots & & & \vdots \\ h_{\mu_n-n+1} & h_{\mu_n-n+2} + h_{\mu_n-n} & \dots & h_{\mu_n} + h_{\mu_n-2n+2} \end{vmatrix}, \tag{1.17}$$

et :

$$O_\mu = \begin{vmatrix} H_{\mu_1} & H_{\mu_1+1} + H_{\mu_1-1} & \dots & H_{\mu_1-1+n} + H_{\mu_1-n+1} \\ H_{\mu_2-1} & H_{\mu_2} + h_{\mu_2-2} & \dots & H_{\mu_2-2+n} + H_{\mu_2-n} \\ \vdots & & & \vdots \\ H_{\mu_n-n+1} & H_{\mu_n-n+2} + H_{\mu_n-n} & \dots & H_{\mu_n} + H_{\mu_n-2n+2} \end{vmatrix}, \tag{1.18}$$

dans laquelle $H_i = h_i - h_{i-2}$.

On peut par exemple donner le développement d'une fonction de Schur symplectique dans la base des fonctions de Schur grâce à la fonction `Sp2s` :

```
ACE> Sp2s(Sp[3,2,1,1]);

                 s[3] - s[3,2] + s[3,2,1,1] - s[3,1,1] - s[1]

                + 2 s[2,1] - s[2,1,1,1] - s[2,2,1] + s[1,1,1]
```

Autre exemple, le produit tensoriel des représentations irréductibles de $Sp(2n)$ se calcule au moyen de la fonction `LRrSp` :

```
ACE> LRrSp(Sp[2,1], Sp[2,1,1]);

   Sp[4, 2, 1] + Sp[4, 1, 1, 1] + Sp[3, 3, 1] + 2 Sp[3, 2, 1, 1] + Sp[3, 2, 2]

      + Sp[3, 1, 1, 1, 1] + Sp[2, 2, 2, 1] + Sp[2, 2, 1, 1, 1] + 3 Sp[2, 2, 1]

      + 3 Sp[2, 1, 1, 1]  + Sp[1, 1, 1, 1, 1]  +  2 Sp[3, 2]   + 3 Sp[3, 1, 1]

      + Sp[4, 1] + 2 Sp[1, 1, 1] + 3 Sp[2, 1] + Sp[3] + Sp[1]
```

1.9 Fonctions symétriques

Nous avons déjà parlé de fonctions symétriques dans les présentations des bibliothèques précédentes, notamment lorsque des fonctions concernaient la théorie des représentations des groupes. La très riche théorie des fonctions symétriques mérite de par ses nombreuses interprétations de disposer de sa propre bibliothèque. La bibliothèque `SYMF` propose ainsi les fonctions classiques de calcul de fonctions symétriques, lesquelles sont utilisées dans plusieurs autres bibliothèques. Nous avons adopté une syntaxe et des notations similaires à celles de la première version de la bibliothèque `SF` de Stembridge. Il convient toute fois de souligner que cette dernière utilisait peu d'algorithmes spécifiques et s'appuyait essentiellement sur le solveur d'équations linéaires de MAPLE, tandis que `SYMF` utilise systématiquement les meilleurs algorithmes combinatoires connus.

Soit $\mathbf{x} = \{x_1,\ x_2,\ \ldots\}$ un ensemble infini d'indéterminées commutatives que nous appellerons un *alphabet*. Les *fonctions symétriques élémentaires* $e_k(\mathbf{x})$ sont définies par la fonction génératrice suivante :

$$\lambda_t(\mathbf{x}) := \sum_{k\geq 0} t^k\, e_k(\mathbf{x}) = \prod_{i\geq 1} (1+x_i t) \quad . \tag{1.19}$$

Les *fonctions symétriques complètes* $h_k(\mathbf{x})$ sont quant à elles définies par :

$$\sigma_t(\mathbf{x}) := \sum_{k\geq 0} t^k\, h_k(\mathbf{x}) = \prod_{i\geq 1} (1-x_i t)^{-1} \quad , \tag{1.20}$$

et les *fonctions symétriques sommes de puissances* $p_k(\mathbf{x})$ par :

$$\psi_t(\mathbf{x}) := \sum_{k\geq 1} t^{k-1}\, p_k(\mathbf{x}) = \sum_{i\geq 1} x_i\,(1-x_i t)^{-1} \quad . \tag{1.21}$$

Ces fonctions génératrices sont liées par les relations suivantes :

$$\psi_t(\mathbf{x}) = \frac{d}{dt}\log\,\sigma_t(\mathbf{x}) = -\frac{d}{dt}\log\,\lambda_{-t}(\mathbf{x}) \quad , \tag{1.22}$$

$$\frac{d}{dt}\,\sigma_t(\mathbf{x}) = \sigma_t(\mathbf{x})\ \psi_t(\mathbf{x}) \quad , \tag{1.23}$$

$$-\frac{d}{dt}\,\lambda_{-t}(\mathbf{x}) = \psi_t(\mathbf{x})\ \lambda_{-t}(\mathbf{x}) \quad , \tag{1.24}$$

la formule (1.24) étant connue sous le nom de formule de Newton. Un théorème fondamental de la théorie est que les $e_k(\mathbf{x})$ sont algébriquement indépendantes. On note *Sym* l'algèbre des fonctions symétriques, *i.e.* l'algèbre engendrée sur $\mathbb{Q}$ par les fonctions (symétriques) élémentaires.

Remarque. Cette algèbre est graduée par le poids $w(e_k) := k$ et la dimension de la composante homogène de poids n est égale à $p(n)$, le nombre de partitions de n (voir formule (7.1), chapitre 7).

Une *partition* est une suite finie décroissante d'entiers positifs (appelés parts de la partition) $\lambda = (\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r)$. Nous écrirons aussi $\lambda = (1^{\alpha_1}\, 2^{\alpha_2}\, \ldots)$, α_m étant le nombre de parts λ_k égales à m. Le *poids* de λ est par définition égal à $|\lambda| = \sum_k \lambda_k$ et sa *longueur* est le nombre de ses parts $\ell(\lambda) = r$.

On associe à chaque partition $\lambda = (1^{\alpha_1}\, 2^{\alpha_2}\, \ldots)$ les éléments suivants :

$$p_\lambda = p_1^{\alpha_1}\, p_2^{\alpha_2}\, \ldots \quad , \quad e_\lambda = e_1^{\alpha_1}\, e_2^{\alpha_2}\, \ldots \quad , \quad h_\lambda = h_1^{\alpha_1}\, h_2^{\alpha_2}\, \ldots \quad .$$

Ils forment des bases de *Sym* et la bibliothèque `SYMF` permet par exemple d'exprimer toute fonction symétrique dans l'une de ces trois bases grâce aux fonctions `Top`, `Toe` et respectivement `Toh` comme illustré par l'exemple suivant :

```
ACE> Toe(h4*p2^2);

           2              2        4              2          3              5
  - 4 e4 e2  + 4 e4 e2 e1  - e4 e1  + 8 e3 e1 e2  - 8 e3 e1  e2 + 2 e3 e1

                      4        3   2          2   4          6     8
          + 4 e2  - 16 e2  e1  + 17 e2  e1  - 7 e2 e1  + e1
```

Pour $\lambda \in \mathbb{Z}^r$, non nécessairement une partition, la *fonction de Schur* s_λ est définie par :

$$s_\lambda = \det\left(h_{\lambda_i - i + j}\right)_{1 \le i,j \le r} \quad , \tag{1.25}$$

où $h_j = 0$ pour $j < 0$. Les fonctions de Schur indicées par des partitions forment une $\mathbb{Q}$-base de *Sym* et la fonction `Tos` exprime toute fonction symétrique comme combinaison linéaire de fonctions de Schur. L'exemple suivant montre une utilisation de cette fonction pour exprimer sur la base des fonctions de Schur le produit d'une fonction de Schur par une fonction complète :

```
ACE> Tos(s[4,3,2,2,1]*h3);

        s[4, 3, 3, 2, 2, 1] + s[4, 4, 2, 2, 2, 1] + s[4, 4, 3, 2, 1, 1]

      + s[4, 4, 3, 2, 2] + s[5, 3, 2, 2, 2, 1] + s[5, 3, 3, 2, 1, 1]

      + s[5, 3, 3, 2, 2] + s[5, 4, 2, 2, 1, 1] + s[5, 4, 2, 2, 2]

      + s[5, 4, 3, 2, 1] + s[6, 3, 2, 2, 1, 1] + s[6, 3, 2, 2, 2]

      + s[6, 3, 3, 2, 1] + s[6, 4, 2, 2, 1] + s[7, 3, 2, 2, 1]
```

On munit *Sym* d'un produit scalaire $\langle .,. \rangle$ pour lequel la base Schur est orthonormale. Les *fonctions monomiales* m_λ sont alors la base adjointe de la base h_λ. La fonction `Tom` permet d'exprimer n'importe quelle fonction symétrique en termes de fonctions monomiales.

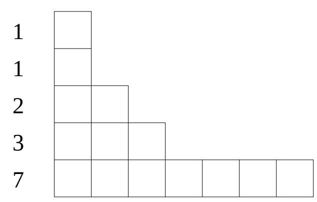
Cette courte introduction montre le côté algébrique de la théorie mais la richesse même des fonctions symétriques tient en fait à la combinatoire sous-jacente prenant en compte les calculs algébriques. Les formules de Newton, Wronski, Muir, Pieri, la règle de Littlewood-Richardson et bien d'autres sont au cœur du système. Par exemple, la formule de Muir, que nous ne détaillerons pas ici, donne sur la base des fonctions de Schur le produit d'une fonction monomiale par une fonction de Schur :

```
ACE> MuirSf(m[2,2,1], s[4,2,2,1]);

                                    s[5, 3, 3, 1, 1, 1] + s[6, 4, 3, 1]
```

```
+ s[5, 4, 4, 1] - s[6, 3, 3, 2]  +  s[6, 4, 2, 2] + s[4, 2, 2, 2, 2, 2]

- s[6, 3, 3, 1, 1] + s[6, 4, 2, 1, 1]  + 3 s[4, 2, 2, 1, 1, 1, 1, 1, 1]

+ 2 s[4, 3, 3, 1, 1, 1, 1]  -  2 s[4, 4, 2, 1, 1, 1, 1] + s[4, 4, 4, 2]

- 2 s[6, 2, 2, 1, 1, 1, 1]  +  s[4, 4, 4, 1, 1]  -  s[6, 2, 2, 2, 1, 1]

+ s[6, 2, 2, 2, 2] -  s[4, 4, 2, 2, 1, 1]  +  s[4, 2, 2, 2, 1, 1, 1, 1]

+ s[4, 4, 2, 2, 2]  + s[5, 2, 2, 1, 1, 1, 1, 1]  -  s[4, 4, 3, 1, 1, 1]

- s[4, 3, 3, 2, 2]  + s[4, 4, 3, 3] -  s[4, 3, 3, 3, 1] - s[5, 3, 3, 3]

- s[6, 3, 2, 1, 1, 1] + s[4, 3, 2, 1, 1, 1, 1, 1] - s[5, 4, 2, 1, 1, 1]

- s[4, 2, 2, 2, 2, 1, 1] + s[4, 3, 3, 2, 1, 1]
```

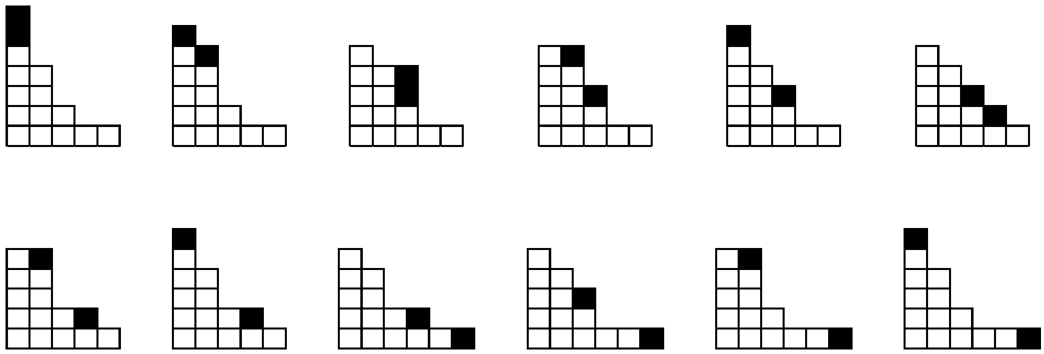
D'autre part, en itérant la formule de Pieri sur une fonction de Schur indicée par une partition vide, nous décrivons un processus donnant l'expression d'un produit de fonctions élémentaires en termes de fonctions de Schur. Pour bien comprendre l'enjeu de l'utilisation de telles formules, nous nous proposons de détailler cet exemple. Le problème initial est d'exprimer dans la base Schur le produit d'une fonction élémentaire par une fonction de Schur, c'est-à-dire essentiellement de décrire les partitions μ apparaissant dans la formule $e_k\, s_\lambda = \sum_\mu s_\mu$.

On peut représenter graphiquement une partition λ au moyen d'un diagramme de boîtes dans le plan, appelé *diagramme de Ferrers* : les lignes de ce diagramme sont numérotées de bas en haut, la i-ième ligne contenant exactement λ_i boîtes. Par exemple, à la partition $\lambda = (7, 3, 2, 1, 1)$ correspond le diagramme de Ferrers suivant :



Cette représentation des partitions est déjà un outil combinatoire puissant. En effet, la formule de Pieri dit exactement la chose suivante : les partitions qui indicent les fonctions de Schur apparaissant dans le développement du produit $e_k\, s_\lambda$ sont exactement celles obtenues par ajout de k boîtes à la périphérie du diagramme de Ferrers de λ avec au plus une seule boîte par ligne et de sorte que le diagramme obtenu soit celui d'une partition.

Par exemple, la multiplication de e_2 par $s_{5,3,2,2,1}$ est prise en compte par l'énumération des diagrammes suivants où figurent en noir les 2 boîtes ajoutées au diagramme de la partition $(5,3,2,2,1)$:



La simple lecture des partitions correspondant à ces diagrammes, c'est-à-dire des nombres de boîtes par ligne, donne les partitions indiçant les fonctions de Schur et par conséquent :

$$\begin{aligned} e_2\, s_{53221} &= s_{5322111} + s_{532221} + s_{53331} + s_{53322} + s_{533211} + s_{54321} \\ &\quad + s_{54222} + s_{542211} + s_{64221} + s_{63321} + s_{63222} + s_{632211} \quad . \end{aligned}$$

Le problème algébrique a donc été remplacé par un problème purement algorithmique, celui de l'énumération d'une certaine classe d'objets combinatoires. Dans le cas de la formule de Pieri, il n'y a aucune difficulté à engendrer ces objets mais la programmation de la règle de Littlewood-Richardson (donnant le produit de deux fonctions de Schur) est par exemple un peu plus délicate.

1.10 Fonctions symétriques non commutatives

Le développement de la toute récente théorie des fonctions symétriques non commutatives a été accompagné de la mise en place, par Bun-Chan-Vorac Ung (Université de Marne-la-Vallée), de la bibliothèque `NCSF`. Partant des quasi-déterminants de Gelfand et Retakh [20, 21], cette théorie a été élaborée [19] de telle sorte que la plupart des applications classiques soit transposable à ce niveau non commutatif. En particulier, l'anneau des caractères du groupe symétrique a pour analogue non commutatif l'algèbre des descentes de Solomon [73].

Les applications sont variées, connectées par exemple aux séries de Lie et aux algèbres de Lie libres (voir [18, 68]). De plus, l'algèbre des fonctions symétriques non commutatives ainsi que son algèbre de Hopf duale, qui est l'algèbre des fonctions quasi-symétriques de Gessel [23, 61], codent la théorie des représentations de la 0-algèbre de Hecke du groupe symétrique [12].

L'*algèbre des fonctions symétriques non commutatives formelles* peut être définie comme l'algèbre associative libre $\mathbf{Sym}=\mathbb{C}\langle\Lambda_1,\Lambda_2,\ldots\rangle$ engendrée par une suite infinie d'indéterminées non commutatives Λ_k appelées *fonctions élémentaires*. Il est d'usage de poser $\Lambda_0 := 1$ et si t désigne une autre indéterminée commutant avec les Λ_k, on introduit les fonctions génératrices suivantes :

$$\lambda(t) \;=\; \sum_{k\geq 0} \Lambda_k\, t^k \quad , \tag{1.26}$$

$$\sigma(t) = \sum_{k\geq 0} S_k\, t^k = \lambda(-t)^{-1} \quad , \tag{1.27}$$

$$\Phi(t) = \sum_{k\geq 1} \frac{1}{k}\Phi_k\, t^k = \log \sigma(t) \quad . \tag{1.28}$$

Les S_k sont appelés *fonctions complètes* et les Φ_k *sommes de puissances de second ordre*. Les *sommes de puissances de premier ordre*, notées Ψ_k, sont les coefficients de la série formelle :

$$\psi(t) = \sum_{k\geq 1} \Psi_k\, t^{k-1} \quad , \tag{1.29}$$

définie par l'équation :

$$\psi(t) = \lambda(-t)\frac{d}{dt}\sigma(t) \quad . \tag{1.30}$$

Les identités précédentes impliquent des relations entre leurs coefficients et l'on a par exemple :

$$\sum_{k=0}^{n}(-1)^k S_{n-k}\Lambda_k = \sum_{k=0}^{n}(-1)^k \Lambda_{n-k} S_k = 0 \quad ,$$
$$\sum_{k=0}^{n-1} S_k \Psi_{n-k} = nS_n \quad ,$$
$$\sum_{k=1}^{n}(-1)^{k-1}\Psi_k \Lambda_{n-k} = n\Lambda_n \quad .$$

Remarque. Dans le cas commutatif, nous avons déjà mentionné l'existence de telles formules dues à Wronski et Newton.

L'algèbre **Sym** est graduée par la fonction de poids w définie par $w(\Lambda_k) = k$ et sa composante homogène de poids n est notée $\mathbf{Sym}_n$. Une *composition* de n est un vecteur $I = (i_1, \ldots, i_k)$ d'entiers positifs tel que $i_1 + \cdots + i_k = n$. Si F_m désigne une fonction symétrique non commutative, on pose pour toute composition $I = (i_1, \ldots, i_k)$ de n :

$$F^I := F_{i_1} F_{i_2} \ldots F_{i_k} \quad . \tag{1.31}$$

Alors, (Λ^I), (S^I), (Φ^I) et (Ψ^I) sont des bases homogènes de **Sym**.

Une composition de n est naturellement codée par un *ruban* composé de n boîtes (voir chapitre 7 pour plus de détail sur les rubans). On considère deux ordres sur l'ensemble des compositions de n. Le premier est l'ordre de *raffinement inverse*, noté $\preceq$ et défini par $I \preceq J$ si et seulement si J est plus *fine* que I. Par exemple, les compositions J de 5 telles que $J \preceq (2,1,2)$ sont $(2,1,2)$, $(3,2)$, $(2,3)$ et (5). Le deuxième ordre est l'ordre *lexicographique inverse* noté $\leq$. Par exemple, $(6) \leq (5,1) \leq (4,2) \leq (4,1,1)$.

On introduit aussi la base des *fonctions de Schur ruban* (R_I). Initialement définie en termes de quasi-déterminants dans [19], on peut aussi les définir par l'un des deux systèmes (équivalents) d'équations suivants :

$$S^I = \sum_{J\preceq I} R_J \quad , \qquad R_I = \sum_{J\preceq I} (-1)^{\ell(I)-\ell(J)} S^J \quad . \tag{1.32}$$

La multiplication des fonctions de Schur rubans est donnée par la formule suivante :

$$R_I R_J = R_{I \triangleright J} + R_{I.J} \quad ,$$

avec $I = (i_1, \ldots, i_r)$, $J = (j_1, \ldots, j_s)$ et où $I \triangleright J = (i_1, \ldots, i_{r-1}, i_r + j_1, j_2, \ldots, j_s)$ et $I.J = (i_1, \ldots, i_r, j_1, \ldots, j_s)$.

La bibliothèque propose naturellement les matrices de transition entre les différentes bases de **Sym**, mais elle fournit aussi les fonctions `ToS`, `ToL`, `ToR`, `ToPh` et `ToPs` donnant les développements de n'importe quel élément de **Sym** dans respectivement la base des produits de fonctions complètes, élémentaires, dans la base des fonctions de Schur rubans, des produits de sommes de puissances de second ou premier ordre. Les algorithmes utilisés pour ces changements de base sont décrits dans [19].

Nous ne nous attarderons pas plus sur cette bibliothèque, cependant nous allons rapidement survoler les quelques développements récents mettant en jeu les fonctions quasi-symétriques.

Soit $\mathbf{x} = \{x_1 < x_2 < x_3 < \cdots\}$ un alphabet infini totalement ordonné et soit $\mathbb{Q}[[\mathbf{x}]]$ l'anneau des séries formelles en ces variables. Une série formelle $F = F(x)$ de $\mathbb{Q}[[\mathbf{x}]]$ est dite quasi-symétrique si pour n'importe quelles suites $y_1 < \cdots < y_k$ et $z_1 < \cdots < z_k$ de $\mathbf{x}$ et n'importe quels entiers $a_1, a_2, \ldots, a_k$, les monômes $y_1^{a_1} \ldots y_k^{a_k}$ et $z_1^{a_1} \ldots z_k^{a_k}$ ont le même coefficient dans F.

Gessel a introduit une base naturelle de **Qsym**, constituée des *fonctions quasi-monomiales* (M_I) définies par :

$$M_I = \sum_{y_1 < y_2 < \cdots < y_k} y_1^{i_1} y_2^{i_2} \ldots y_k^{i_k} \quad ,$$

où $I = (i_1, i_2, \ldots, i_k)$ est une composition de n. Les *fonctions quasi-rubans* (F_I) ont aussi été introduite par Gessel et constituent une autre base intéressante :

$$F_I = \sum_{I \preceq J} M_J \quad .$$

Comme dans la partie concernant **Sym**, la bibliothèque propose les fonctions `ToM` et `ToF` permettant d'exprimer n'importe quelle fonction quasi-symétrique dans les deux bases précédemment introduites. De même, l'utilisateur dispose de deux fonctions donnant les matrices de transitions entre ces deux bases. Ce sont les fonctions `F2MMat` et `M2FMat` qui réalisent ces deux calculs comme l'illustre l'exemple suivant pour la composante homogène de degré 3 :

✂ ....

```
ACE> F2MMat(3);

                              [ 1   1   1   1 ]
                              [               ]
                              [ 0   1   0   1 ]
                              [               ]
                              [ 0   0   1   1 ]
                              [               ]
                              [ 0   0   0   1 ]
```

```
ACE> M2FMat(3);

                              [ 1  -1  -1   1 ]
                              [               ]
                              [ 0   1   0  -1 ]
                              [               ]
                              [ 0   0   1  -1 ]
                              [               ]
                              [ 0   0   0   1 ]
```

En particulier les fonctions symétriques sont quasi-symétriques. Selon Gessel [23], si f est une fonction symétrique, alors :

$$f = \sum_I \langle f, r_I \rangle F_I \quad ,$$

où r_I est l'image commutative de la fonction de Schur ruban R_I et $\langle .,. \rangle$ est le produit scalaire de l'algèbre des fonctions symétriques dont nous avons déjà parlé et qui est défini par :

$$\langle s_\lambda, s_\mu \rangle = \delta_{\lambda\mu} \quad .$$

La bibliothèque fournit la fonction `Sf2Qsf` qui exprime n'importe quelle fonction symétrique en termes de fonctions quasi-symétriques en privilégiant la base des (F_I) pour donner son résultat :

```
ACE> Sf2Qsf(s[3,2]);

            F[3, 2] + F[2, 3] + F[2, 2, 1] + F[1, 3, 1] + F[1, 2, 2]
```

La matrice d'ordre $p(n) \times 2^{n-1}$ dont les lignes donnent les coefficients du développement de la fonction de Schur s_λ dans la base des (F_I) peut être interprétée comme la matrice de décomposition de la 0-algèbre de Hecke $\mathcal{H}_n(0)$ (voir [4]).

1.11 Groupes hyperoctaédraux

Nous avons récemment mis en place la bibliothèque `BN` de calcul sur les groupes hyperoctaédraux car c'est une extension naturelle de la bibliothèque relative aux groupes symétriques. En fait, le groupe hyperoctaédral B_n contient comme sous-groupe $\mathfrak{S}_{n-1}$ et se plonge dans $\mathfrak{S}_{2n}$. Il fait partie, tout comme le groupe symétrique, des groupes de Coxeter finis.

On note B_n le groupe de Coxeter engendré par $s_0, \ldots s_{n-1}$. Les générateurs $s_1, \ldots, s_{n-1}$ sont ceux du groupe symétrique $\mathfrak{S}_n$, et s_0 satisfait les relations suivantes :

$$\begin{cases} s_0^2 = 1 & , \\ s_0 s_1 s_0 s_1 = s_1 s_0 s_1 s_0 & , \\ s_0 s_i = s_i s_0 & , \quad i > 1 \quad . \end{cases}$$

Un *code* (au sens de B_n) est un vecteur de longueur n, plus petit composante à composante que le vecteur $[1, 3, 5, \ldots, 2n-1]$. L'élément *maximal* de B_n qui correspond au code $[1, 3, 5, \ldots, 2n-1]$ a une décomposition réduite canonique :

$$(0)\,(101)\,(21012)\ \ldots\ (n-1\ldots 0\ldots n-1) \quad ,$$

factorisée en blocs du type :

$$(k\ldots 101\ldots k) \quad ,$$

dans lequel i dénote le générateur s_i.

Tout élément w de B_n possède alors une décomposition réduite canonique qui consiste en un produit de facteurs gauches des facteurs canoniques précédents, ces facteurs étant caractérisés par leurs longueurs $c_1, \ldots, c_n$ (le vecteur $[c_1, \ldots, c_n]$ étant le code de w). Cette décomposition réduite se caractérise aussi comme étant la plus petite pour l'ordre lexicographique. Par exemple, la décomposition réduite (0) (10) () (32101) correspond au code $[1, 2, 0, 5]$. On peut obtenir ce type de décomposition réduite grâce à la fonction `BnCode2Rd`. La *longueur* d'un élément w, donnée par la fonction `BnCode2Length`, est la longueur minimum d'une expression de w comme produit de générateurs. C'est aussi la somme de toutes les entrées de son code, comme dans le cas du groupe symétrique.

Comme dans le cas du groupe symétrique, la décomposition d'un élément du groupe hyperoctaédral en un produit de générateurs simples n'est pas unique et la liste de toutes les décompositions réduites d'un élément donné par son code est obtenue grâce à la fonction `BnCode2ListRd` comme le montre l'exemple suivant :

```
ACE> BnCode2ListRd([1,3,3]);

    [[1, 0, 1, 2, 0, 1, 0], [1, 0, 1, 0, 2, 1, 0], [0, 1, 0, 1, 2, 1, 0],

     [0, 1, 2, 0, 1, 2, 0], [0, 1, 0, 2, 1, 2, 0], [0, 1, 2, 0, 1, 0, 2],

     [0, 1, 0, 2, 1, 0, 2]]
```

alors que la fonction `BnCode2NbRd` retourne simplement le nombre de ces expressions :

```
ACE> BnCode2NbRd([1,3,5,7]);

                                   24024
```

La combinatoire du nombre de décompositions réduites pour les éléments de B_n est moins développée que celle de $\mathfrak{S}_n$. Néanmoins, voici le nombre de décompositions réduites pour l'élément maximum de B_1 à B_6 (la loi générale est claire !) :

```
ACE> seq(BnCode2NbRd(BnLastCode(i)), i=1..6);

                 1, 2, 42, 24024, 701149020, 1671643033734960
```

La fonction `BnListCode` retourne les $2^n n!$ codes des éléments de B_n. De plus, plusieurs fonctions sont disponibles pour engendrer pas à pas les codes de B_n : `BnFirstCode`(n) retourne le premier code de B_n, `BnLastCode`(n) le dernier code, `BnPrevCode` et `BnNextCode` donnant respectivement le code précédent et le code suivant un code donné, le tout pour l'ordre lexicographique.

Comme dans le cas du groupe symétrique, on peut engendrer les codes des éléments plus petits qu'un élément donné pour l'ordre de Bruhat :

```
ACE> BnInterval([1,3,3]);

 [[1, 2, 4], [1, 1, 4], [1, 2, 3], [1, 2, 2], [1, 1, 3], [1, 1, 2], [1, 2, 1],

  [1, 1, 1], [0, 2, 4], [0, 1, 4], [0, 2, 3], [0, 2, 2], [0, 1, 3], [0, 1, 2],

  [0, 2, 1], [0, 1, 1], [1, 0, 4], [0, 0, 4], [1, 0, 3], [1, 0, 2], [0, 0, 3],

  [0, 0, 2], [1, 0, 1], [0, 0, 1], [1, 3, 0], [0, 3, 0], [1, 2, 0], [1, 1, 0],

  [0, 2, 0], [0, 1, 0], [1, 0, 0], [0, 0, 0]]
```

Les opérations de groupe sont définies sur les éléments donnés par leurs codes. La fonction `BnMultCode` réalise la multiplication de deux éléments de B_n alors que la fonction `BnInvCode` inverse un élément donné.

Il y a d'autres manières de coder les éléments de B_n. Nous en donnons deux autres qui sont plus fréquemment choisies : les permutations signées et les permutations palindromes. Notre codage présente cependant l'avantage d'être plus compact et de fournir instantanément une décomposition réduite de l'élément codé.

Tout d'abord, on peut plonger B_n dans le groupe symétrique $\mathfrak{S}_{2n}$. Cela revient à interpréter les générateurs (alors donnés par la fonction `BnTranspoPalin`) comme suit :

$$\begin{cases} s_0 &=& t_n \quad , \\ s_1 &=& t_{n-1}t_{n+1} \quad , \\ \vdots & \vdots & \vdots \\ s_{n-1} &=& t_1 t_{2n-1} \quad , \end{cases}$$

où les t_i sont les transpositions élémentaires engendrant $\mathfrak{S}_{2n}$. L'image de B_n consiste alors en les permutations $v := [v_1, \ldots, v_{2n}]$ dites *palindromes*—par un léger abus de langage— que nous appelons plus simplement `Palin` dans la bibliothèque et qui ont la propriété suivante :

$$v_1 + v_{2n} = v_2 + v_{2n-1} = \cdots = v_n + v_{n+1} = 2n + 1 \quad .$$

Nous fournissons les différents décodages `BnCode2Palin` et `BnPalin2Code` et bien entendu l'utilisateur pourra manipuler les permutations palindromes grâce aux outils disponibles dans la bibliothèque `SG`.

Une autre description de B_n consiste à regarder le groupe comme l'ensemble des permutations signées de $\{1,\ldots,n\}$, permutations que nous appelons `Bar` dans la bibliothèque. Soit $u := [u_1,\ldots,u_n]$, alors u est une permutation signée si $[|u_1|,\ldots,|u_n|]$ est une permutation de $\{1,\ldots,n\}$. L'action du générateur $s_i, i > 0$ (sur sa gauche) sur une permutation signée $u := [u_1,\ldots,u_n]$ consiste en l'échange de u_i et u_{i+1} :

$$[u_1,\ldots,u_i,u_{i+1},\ldots,u_n]\,s_i \;=\; [u_1,\ldots,u_{i+1},u_i,\ldots,u_n]\;, \quad i>0 \quad .$$

Le générateur s_0 transforme quant à lui u_1 en $-u_1$:

$$[u_1,\ldots,u_n]\,s_0 \;=\; [-u_1,\ldots,u_n] \quad .$$

Symétriquement, le produit $s_i[u_1,\ldots,u_n]$ consiste à échanger les valeurs i et $i+1$, pour $i > 0$ et échanger 1 et -1 pour $i = 0$. Comme précédemment, nous donnons les fonctions réalisant les différentes conversions, c'est-à-dire `BnCode2Bar` et `BnBar2Code` mais cette fois nous fournissons aussi la fonction de multiplication de deux permutations signées : `BnMultBar`.

Les permutations signées (`Bar`) peuvent, à la manière des permutations du groupe symétrique, être représentées par des matrices de permutation, quelques entrées égales à 1 étant changées en -1. Pour faire le lien avec la bibliothèque `SG`, nous donnons ainsi les deux fonctions `BnBar2Mat` et `BnMat2Bar` calculant ou décodant ces matrices.

1.12 Algèbres liées aux groupes hyperoctaédraux

Lorsque nous avons mis en place la bibliothèque `SG` pour le groupe symétrique, nous lui avons rapidement adjoint plusieurs bibliothèques permettant de calculer dans différentes algèbres directement liées au groupe symétrique et de faire agir ces algèbres sur l'anneau des polynômes. Nous avons fait de même avec la bibliothèque `BN` et nous lui avons adjoint la bibliothèque `BNA` qui regroupe les fonctions de calcul dans trois algèbres différentes ainsi que leurs actions respectives sur l'anneau des polynômes. Le tableau suivant résume les préfixes des noms de fonctions correspondant à ces trois algèbres :

Préfixe	**Algèbre**
`BnGa`	pour l'algèbre de groupe de B_n ;
`BnNca`	pour l'algèbre nilCoxeter de B_n ;
`BnIdca`	pour l'algèbre idCoxeter de B_n.

Les éléments de ces algèbres sont représentés par des sommes d'éléments de B_n (donnés par leurs codes) avec n'importe quel type de coefficients. L'addition et la soustraction, communes à l'ensemble de ces algèbres, font bien entendu partie de la bibliothèque via les fonctions `BnaAdd` et `BnaMinus`. Pour chacune de ces algèbres, nous donnons une fonction de multiplication de deux éléments. Le tableau suivant résume à nouveau les trois fonctions existantes en décrivant succintement la multiplication élémentaire sous-jacente :

Fonction	Objet
`BnGaMult`	utilise la multiplication dans B_n ;
`BnNcaMult`	utilise la multiplication où les relations $s_i^2 = 1, i \geq 0$ ont été remplacées par $\partial_i^B \partial_i^B = 0$;
`BnIdcaMult`	utilise la multiplication où les relations $s_i^2 = 1, i \geq 0$ ont été remplacées par $\pi_i^B \pi_i^B = -\pi_i^B$.

L'algèbre de groupe (qui correspond au préfixe `BnGa`) agit sur l'anneau des polynômes en $x_1, \ldots, x_n$ (par défaut). Les s_i, $i = 1, \ldots, n-1$ sont les générateurs usuels de $\mathfrak{S}_n$, *i.e.* s_i transpose x_i et x_{i+1} alors que s_0 change x_1 en $-x_1$. La fonction `BnGaOnPol` réalise effectivement cette action.

L'algèbre nilCoxeter (`BnNca`) agit aussi sur l'anneau des polynômes en $x_1, \ldots, x_n$ (par défaut) grâce cette fois à la fonction `BnNcaOnPol`. Les ∂_i^B, $i = 1, \ldots, n-1$ sont les différences divisées usuelles, *i.e.* ∂_i^B agit par :

$$p \quad \rightarrow \quad \frac{p - s_i(p)}{x_i - x_{i+1}} \quad .$$

La différence divisée supplémentaire ∂_0^B est quant à elle définie par :

$$p \quad \rightarrow \quad \frac{p - s_0(p)}{2x_1} \quad .$$

De même pour l'algèbre idCoxeter (`BnIdca`) qui agit cette fois sur l'anneau des polynômes (de Laurent) en $x_1, \ldots, x_n$ (par défaut). C'est la fonction `BnIdcaOnPol` qui réalise cette action. Les π_i^B, $i = 1, \ldots, n-1$ sont les générateurs usuels et par suite agissent par :

$$p \quad \rightarrow \quad \frac{p - s_i(p)}{x_i - x_{i+1}} x_{i+1} \quad .$$

Le générateur supplémentaire π_0^B est quant à lui défini par :

$$p \quad \rightarrow \quad \frac{x_1 p - \frac{1}{x_1} p'}{1/x_1 - x_1} \quad ,$$

où p' est obtenu à partir de p en changeant x_1 en $1/x_1$. Par exemple :

$$\pi_0^B(x_1^k) = -(x_1^k + x_1^{k-2} + \cdots + x_1^{-k+2} + x_1^{-k}) \quad ,$$

résultat que l'on vérifie avec la bibliothèque :

```
ACE> expand(BnIdcaOnPol([1], x1^6));

                  6      4      2          1      1      1
              - x1  - x1  - x1  - 1 - --- - --- - ---
                                            2      4      6
                                          x1     x1     x1
```

Remarque. Les éléments de ces algèbres peuvent aussi être décodés en termes de permutations signées ou permutations palindromes grâce aux deux fonctions `BnaCode2Bar` et `BnaCode2Palin`, utiles aux récalcitrant du codage des éléments de B_n par des codes.

Dans les deux types d'algèbre, de groupe et nilCoxeter de B_n, nous donnons une base de Yang-Baxter calculée par les fonctions **BnGaYang** et **BnNcaYang**. Ces bases, indicées par des codes, dépendent de paramètres en $x_1, \ldots, x_n$ (par défaut), lesquels peuvent être spécialisés en $1, \ldots, n$ par l'option **'num'** :

```
ACE> BnGaYang([1,3]);

                                      3        2
   - 2 (x2 + x1) x2 [0, 2] + (2 x1  - 2 x2  x1 - 2 x1 - 2 x2) [1, 0]

                                                     2     2
   - 4 (x1 - x2) (x2 + x1) x1 x2 [1, 3] + (- x1  + x2  + 1 + 4 x2 x1) [0, 0]

            2                              2             2
   + (- 2 x2  - 2 x2 x1) [1, 1] + (- 4 x2 x1  + 2 x2 + 4 x2  x1) [0, 1]

   + 2 (x1 - x2) (x2 + x1) x2 [0, 3] + 4 (x2 + x1) x1 x2 [1, 2]

ACE> BnGaYang([1,3], 'num');

   - 12 [0, 2] - 12 [1, 0] + 12 [0, 0] - 12 [1, 1] + 12 [0, 1] + 24 [1, 2]

   - 12 [0, 3] + 24 [1, 3]

ACE> BnNcaYang([1,3]);

   - 2 (x2 + x1) x2 [0, 2] + (- 2 x1 - 2 x2) [1, 0]

                                                                2
   - 4 (x1 - x2) (x2 + x1) x1 x2 [1, 3] + [0, 0] + (- 2 x2  - 2 x2 x1) [1, 1]

   + 2 x2 [0, 1] + 2 (x1 - x2) (x2 + x1) x2 [0, 3] + 4 (x2 + x1) x1 x2 [1, 2]
```

1.13 Tableaux de Young

Les *tableaux de Young* sont des structures planaires. Ils sont obtenus par remplissage par des entiers de diagrammes de Ferrers de sorte que les lignes (lues de gauche à droite) soient des suites croissantes au sens large et les colonnes (lues de bas en haut), strictement croissantes. Un tableau est dit *standard* lorsque tous les nombres utilisés sont les entiers $1, 2, 3, \ldots$ sans répétition. Parmi leurs nombreuses interprétations, les tableaux codent par exemple des bases linéaires des représentations irréductibles du groupe symétrique ou du groupe linéaire mais leurs multiples propriétés combinatoires justifient la réalisation d'une bibliothèque spéciale. Cette partie détaille la bibliothèque `TAB` qui contient les principaux algorithmes combinatoires sur les tableaux.

Nous représentons un tableau par la suite des lignes qui le composent. C'est ainsi que le tableau suivant :

3	5	
2	4	
1	1	3

sera codé dans la bibliothèque par la liste : `[ [3,5] ,[2,4], [1,1,3] ]`. On retrouve à partir de cette représentation la structure planaire en utilisant la fonction `Tab2Mat` comme illustré par l'exemple suivant :

```
ACE> Tab:=[ [3,5] ,[2,4], [1,1,3] ];

                        Tab := [[3, 5], [2, 4], [1, 1, 3]]

ACE> Tab2Mat(Tab);

                                   [ 3  5    ]
                                   [         ]
                                   [ 2  4    ]
                                   [         ]
                                   [ 1  1  3 ]
```

Un tableau peut être considéré comme un *mot* obtenu par concaténation de ses lignes (c'est la fonction `Tab2Word` qui réalise cette opération). A partir de ce mot, on reconstruit sans ambiguïté le tableau correspondant en coupant le mot à chaque descente.

Plus généralement, Schensted [70] a décrit un algorithme (un processus d'insertion) qui transforme n'importe quel mot en un tableau : si `Tab` est un tableau obtenu par cet algorithme à partir du mot w, le tableau correspondant au mot wx est obtenu par la commande `Bump(Tab,` x`)` :

```
ACE> Bump(Tab, [7,8,6,7]);

                     [[3, 5], [2, 4, 7, 8], [1, 1, 3, 6, 7]]

ACE> Tab2Mat(");

                                [ 3  5           ]
                                [                ]
                                [ 2  4  7  8     ]
                                [                ]
                                [ 1  1  3  6  7  ]
```

Deux mots qui donnent le même tableau sont dits *plaxiquement* équivalents. Knuth [39] a montré que l'équivalence plaxique se déduisait des équivalences suivantes n'impliquant que des mots de longueur 3 dans lesquels $i < j < k$:

$$kij \equiv ikj, \quad jki \equiv jik, \quad jii \equiv iji, \quad jji \equiv jij \quad .$$

En poussant, grâce au "jeu de taquin", un tableau planaire dans le coin nord-est, on obtient un *contre-tableau* que nous codons dans la bibliothèque par la suite de ces colonnes, lues de gauche à droite et de haut en bas. Nous fournissons les fonctions `CTab2Word` et `CTab2Mat` qui sont exactement les analogues des fonctions pour les tableaux.

Nous avons aussi inclus un algorithme (réalisé par la fonction `Word2CTab`) construisant à partir de n'importe quel mot un contre-tableau, l'étape élémentaire qui correspond à l'insertion d'une lettre étant donnée par la fonction `BumpC`. L'exemple suivant montre quelques utilisations de ces différentes fonctions :

```
ACE> CTab:=Word2CTab([4,2,3,4,1,2,3,2]);

                          CTab := [[2, 1], [4, 3, 2], [4, 3, 2]]

ACE> CTab2Mat(CTab);

                                        [ 2  4  4 ]
                                        [         ]
                                        [ 1  3  3 ]
                                        [         ]
                                        [    2  2 ]

ACE> CTab2Mat( BumpC(CTab, [7,8,6,7]) );

                                     [ 2  4  4  7  8 ]
                                     [               ]
                                     [ 1  3  3  6  7 ]
                                     [               ]
                                     [          2  2 ]
```

Une autre manière de coder les tableaux standards vient des *mots de Yamanouchi*, c'est-à-dire des mots w tels que pour chaque factorisation $w = u\,v$, $|v|_1 \geq |v|_2 \geq |v|_3 \geq \cdots$, où $|v|_i$ dénote le nombre de lettres i apparaissant dans v. Le codage dit simplement dans quelles lignes du tableau standard se trouvent les lettres $n, n-1, \ldots, 2, 1$. Nous donnons la fonction inverse, `Yama2StdTab`, qui permet de passer d'un mot de Yamanouchi à un tableau standard.

La *forme* d'un tableau est la liste des longueurs de ses lignes successives et cette partition est donnée par la fonction `Tab2Part`. La fonction `ListTab` retourne la liste de tous les tableaux d'une forme donnée et d'une évaluation donnée, comme le montre l'exemple suivant où nous listons tous les tableaux de forme $(3, 2)$ ayant une fois les lettres 2, 5, 9 et deux fois la lettre 6 :

```
ACE> ListTab([3,2], [0,1,0,0,1,2,0,0,1]);

          [[[6, 9], [2, 5, 6]], [[5, 9], [2, 6, 6]], [[6, 6], [2, 5, 9]]]

ACE> map(Tab2Mat, ");

                        [ 6  9    ]  [ 5  9    ]  [ 6  6    ]
                       [[         ], [         ], [         ]]
                        [ 2  5  6 ]  [ 2  6  6 ]  [ 2  5  9 ]
```

Etant donnée une partition λ, la fonction `Part2ListStdTab` énumère tous les tableaux standards de forme λ :

```
ACE> Part2ListStdTab([3,1]);

               [[[2], [1, 3, 4]], [[3], [1, 2, 4]], [[4], [1, 2, 3]]]

ACE> map(Tab2Mat, ");

                        [ 2       ]  [ 3       ]  [ 4       ]
                       [[         ], [         ], [         ]]
                        [ 1  3  4 ]  [ 1  2  4 ]  [ 1  2  3 ]
```

L'énumération équivalente des mots de Yamanouchi est donnée par la fonction `ListYama`. Cette fonction calcule les mots correspondant à une forme donnée, ces mots étant exactement les mots envoyés sur les tableaux $[\ldots, [3\ldots3], [2\ldots2], [1\ldots1]]$ par l'algorithme de Schensted. L'exemple suivant reprend cette propriété :

```
ACE> ListYama([4,2,1]);

     [[1, 2, 1, 4, 3, 2, 1], [2, 1, 1, 4, 3, 2, 1], [2, 1, 4, 1, 3, 2, 1],

      [2, 4, 1, 1, 3, 2, 1], [4, 2, 1, 1, 3, 2, 1], [4, 2, 1, 3, 1, 2, 1],

      [4, 2, 3, 1, 1, 2, 1], [4, 3, 2, 1, 1, 2, 1], [4, 3, 2, 1, 2, 1, 1],

      [4, 3, 2, 2, 1, 1, 1], [4, 2, 3, 1, 2, 1, 1], [4, 2, 3, 2, 1, 1, 1],

      [4, 2, 1, 3, 2, 1, 1], [2, 4, 1, 3, 1, 2, 1], [2, 4, 3, 1, 1, 2, 1],

      [2, 4, 3, 1, 2, 1, 1], [2, 4, 3, 2, 1, 1, 1], [2, 4, 1, 3, 2, 1, 1],

      [2, 1, 4, 3, 1, 2, 1], [2, 1, 4, 3, 2, 1, 1], [1, 2, 4, 1, 3, 2, 1],

      [1, 4, 2, 1, 3, 2, 1], [4, 1, 2, 1, 3, 2, 1], [4, 1, 2, 3, 1, 2, 1],
```

```
    [4, 1, 3, 2, 1, 2, 1], [4, 3, 1, 2, 1, 2, 1], [4, 3, 1, 2, 2, 1, 1],

    [4, 1, 3, 2, 2, 1, 1], [4, 1, 2, 3, 2, 1, 1], [1, 4, 2, 3, 1, 2, 1],

    [1, 4, 3, 2, 1, 2, 1], [1, 4, 3, 2, 2, 1, 1], [1, 4, 2, 3, 2, 1, 1],

    [1, 2, 4, 3, 1, 2, 1], [1, 2, 4, 3, 2, 1, 1]]

ACE> map(Word2Tab, "):
ACE> Tab2Mat( op( {op(")} ) );

                                 [ 4       ]
                                 [         ]
                                 [ 3       ]
                                 [         ]
                                 [ 2  2    ]
                                 [         ]
                                 [ 1  1  1 ]
```

Un tableau t donne un mot en les lettres non commutatives $1, 2, \ldots$, mais aussi un monôme noté x^t en remplaçant chaque lettre i par la variable x_i. Grâce à cette transformation, la somme de tous les tableaux d'une forme donnée λ donne un polynôme $\sum x^t$ qui est exactement la fonction de Schur d'indice la forme du tableau.

En fait, il existe une action du groupe symétrique sur l'algèbre libre, compatible avec la congruence plaxique, l'image d'un tableau étant un tableau de même forme. La somme des tableaux d'une forme donnée est un invariant pour cette action. Dans la bibliothèque `TAB`, cette action est donnée par la fonction `PermOnWord`. La projection de l'algèbre libre sur l'anneau des polynômes envoie cet invariant sur une fonction de Schur.

Finalement, à chaque mot on associe deux vecteurs d'entiers qui sont son *vecteur de charge* (calculé par la fonction `Word2Charge`) et son *vecteur de cocharge* [71] (que l'on peut calculer par la fonction `Word2CoCharge`) :

```
ACE> Word2Charge([3,2,2,1,4,1,1,3], 'vector');

                              [0, 0, 0, 0, 1, 0, 0, 1]

ACE> Word2CoCharge([3,2,2,1,4,1,1,3], 'vector');

                              [2, 1, 1, 0, 2, 0, 0, 1]
```

Ces deux vecteurs sont compatibles avec l'algorithme de Schensted et les relations plaxiques, ainsi qu'avec l'action cyclique de permutation sur les mots. La somme des entrées du vecteur de cocharge est un entier qui est égal à la hauteur d'un tableau lorsque l'on munit l'ensemble des tableaux d'une structure d'ensemble partiellement ordonné par l'opération de suppression de la première lettre d'un tableau et d'insertion de cette lettre par la droite (lorsque le tableau n'est pas une ligne) :

$$t = [[x\,u][v]\cdots[w]] \quad \longrightarrow \quad t' = \texttt{Word2Tab}([u,v,\ldots,w,x]) \quad .$$

```
ACE> lt:=ListTab([3,2,1], [1,2,2,1]);

        lt := [[[4], [3, 3], [1, 2, 2]], [[4], [2, 3], [1, 2, 3]],

               [[3], [2, 4], [1, 2, 3]], [[3], [2, 3], [1, 2, 4]]]

ACE> map(Tab2Mat, lt);

              [ 4       ]  [ 4       ]  [ 3       ]  [ 3       ]
              [         ]  [         ]  [         ]  [         ]
             [[ 3  3    ], [ 2  3    ], [ 2  4    ], [ 2  3    ]]
              [         ]  [         ]  [         ]  [         ]
              [ 1  2  2 ]  [ 1  2  3 ]  [ 1  2  3 ]  [ 1  2  4 ]

ACE> map(Tab2Word, lt);

                [[4, 3, 3, 1, 2, 2], [4, 2, 3, 1, 2, 3],
                 [3, 2, 4, 1, 2, 3], [3, 2, 3, 1, 2, 4]]

ACE> map(Word2Charge, ");

                             [3, 2, 2, 1]
```

Par ailleurs, la somme :

$$\sum_{|\lambda|=|\mu|} \Big(\sum_{t\in \mathrm{Tab}(\lambda,\mu)} q^{c(t)} \Big) s_\lambda \quad ,$$

dans laquelle $\mathrm{Tab}(\lambda,\mu)$ dénote l'ensemble de tous les tableaux d'une forme donnée λ et d'une évaluation donnée μ et $c(t)$ désigne la somme des entrées du vecteur de charge de t, est le *polynôme de Hall-Littlewood* Q'_μ.

1.14 Polynômes de Schubert

Rappelons qu'à chaque permutation $\mu \in \mathfrak{S}_n$ est associée une différence divisée ∂_μ, qui peut être calculée en prenant une décomposition réduite $s_{i_1}\ldots s_{i_k}$ de μ et en posant :

$$\partial_\mu := \partial_{i_1}\ldots\partial_{i_k} \quad . \tag{1.33}$$

Nous avons déjà mentionné un exemple d'utilisation de ces opérateurs. Soit $p \in \mathbb{K}[x_1]$ un polynôme en l'unique variable x_1 de degré inférieur à n et soit $\{y_1,\ldots,y_{n-1}\}$ un ensemble d'éléments de $\mathbb{K}$. La formule d'interpolation de Newton s'écrit :

$$p(x_1) = p(y_1) + \sum_{i=1}^{n-1} (x_1-y_1)\ldots(x_1-y_i)\ \partial_i\ldots\partial_1(p) \quad , \tag{1.34}$$

dans laquelle les différences divisées agissent sur l'alphabet $\{y_1, \ldots, y_{n-1}\}$. Nous verrons un peu plus loin une extension de cette formule au cas de plusieurs variables.

Remarque. La propriété $\partial_i^2 = 0$ implique que si $s_{i_1} \ldots s_{i_k}$ n'est pas une décomposition réduite alors $\partial_{i_1} \ldots \partial_{i_k} = 0$.

Pour toute permutation $\mu \in \mathfrak{S}_n$, le *polynôme de Schubert (simple)* [49] $\mathbf{X}_\mu$ est défini par :

$$\mathbf{X}_\mu := \partial_{\mu^{-1}\omega}(x^\delta) \quad ,$$

où ω dénote l'élément maximal(3) de $\mathfrak{S}_n$, μ^{-1} l'inverse de μ et $x^\delta := x_1^{n-1} x_2^{n-2} \ldots x_n^0$ le "monôme escalier". Les polynômes de Schubert sont parfois indicés par des suites $\mathbf{i} = (i_1, \ldots, i_n) \subset \mathbb{N}^n$ en prenant les codes de μ plutôt que les permutations elles-mêmes. On note alors $\mathbf{Y_i}$ le polynôme de Schubert correspondant à la suite $\mathbf{i} \subset \mathbb{N}^n$ pour éviter toute confusion.

La définition précédente entraîne le calcul suivant de $\mathbf{X}_\mu$ pour $\mu = [1, 4, 2, 3]$. On a $\omega = [4, 3, 2, 1]$ et $\mu^{-1}\omega = s_1 s_2 s_3 s_2$, d'où :

$$\begin{aligned}
\mathbf{X}_{1,4,2,3} &= \partial_{2,4,3,1}(x^\delta) = \partial_1\partial_2\partial_3\partial_2(x_1^3 x_2^2 x_3) \\
&= \partial_1\partial_3\partial_2\partial_3(x_1^3 x_2^2 x_3) = \partial_1\partial_3\partial_2(x_1^3 x_2^2) \\
&= \partial_1\partial_3(x_1^3 x_2 + x_1^3 x_3) = \partial_1(x_1^3) = x_1^2 + x_1 x_2 + x_2^2 \quad .
\end{aligned}$$

La fonction `TableX` de la bibliothèque `SP` calcule tous les polynômes de Schubert correspondant aux permutations d'un groupe symétrique donné. L'exemple suivant illustre ce calcul pour $\mathfrak{S}_3$, la fonction retournant une table :

```
ACE> TableX(3);

                        table([
                          [1, 2, 3] = 1
                          [2, 1, 3] = x1
                          [1, 3, 2] = x1 + x2
                          [2, 3, 1] = x1 x2
                                         2
                          [3, 1, 2] = x1
                                         2
                          [3, 2, 1] = x1  x2
                        ])
```

Les polynômes de Schubert $\mathbf{X}_\mu$, $\mu \in \mathfrak{S}_n$ forment une $\mathbb{Z}$-base du sous-espace $\mathbb{Z}[x_1, \ldots, x_n]$ engendré par les monômes $x^\alpha, \alpha \subseteq (n-1, n-2, \ldots, 0)$ et par conséquent les polynômes de Schubert $\mathbf{X}_\mu$, $\mu \in \mathfrak{S}_\infty$ forment une $\mathbb{Z}$-base de l'anneau des polynômes $\mathbb{Z}[x_1, x_2, \ldots]$ en une infinité de variables. La fonction `ToX` développe toute expression dans la base des polynômes de Schubert. Par ailleurs, la fonction `x2X` convertit tout polynôme en les x_i dans la base Schubert alors que la fonction `X2x` réalise l'opération inverse, comme le montre l'exemple suivant :

(3) C'est-à-dire la permutation ayant le nombre maximal d'inversions.

```
ACE> x2X(x1^5 + x1*x2^3*x3);

      X[2, 5, 3, 1, 4] - X[3, 4, 2, 1] - X[4, 2, 3, 1] + X[6, 1, 2, 3, 4, 5]

ACE> X2x(");
                                  5       3
                                x1  + x1 x2  x3
```

La structure multiplicative de $Sym[x_1, \ldots, x_n]$, dans la base linéaire des fonctions de Schur, est déterminée par la formule de Pieri qui exprime dans la base Schur le produit $h_i s_\lambda$ (ou $e_i s_\lambda$), $i \geq 0$, λ étant une partition et où h_i (resp. e_i) dénote la i-ième fonction symétrique complète (resp. élémentaire). Similairement, la structure multiplicative de $\mathbb{Z}[x_1, x_2, \ldots]$ dans la base linéaire des polynômes de Schubert, est déterminée par le produit des polynômes de Schubert par les x_i. Plus précisément, on a la formule suivante due à Monk [64] ([57, (4.15')]) :

$$x_r \mathbf{X}_\mu = \sum_{\nu = \mu\, t_{ri},\, i > r,\, l(\nu) = l(\mu)+1} \mathbf{X}_\nu - \sum_{\eta = \mu\, t_{ir},\, i < r,\, l(\eta) = l(\mu)+1} \mathbf{X}_\eta \quad , \tag{1.35}$$

où t_{ij} dénote la transposition qui échange i et j. La fonction `Monk` réalise cette opération. Par exemple, le calcul de $x_4 \mathbf{X}_{3,6,2,4,8,7,1,5}$ se présente ainsi :

```
ACE> Monk(4, X[3,6,2,4,8,7,1,5]);

            - X[3, 6, 4, 2, 8, 7, 1, 5] - X[4, 6, 2, 3, 8, 7, 1, 5]

            + X[3, 6, 2, 8, 4, 7, 1, 5] + X[3, 6, 2, 7, 8, 4, 1, 5]

            + X[3, 6, 2, 5, 8, 7, 1, 4]
```

Une *formule de Pieri* [49] plus générale donne l'expression du produit $h_i \mathbf{X}_\mu$ (resp. $e_i \mathbf{X}_\mu$) dans la base Schubert. Nous avons cependant pris comme algorithme de base la formule de Monk. Notons que l'on peut choisir r de sorte qu'il n'y ait qu'un seul terme positif dans le produit $x_r \mathbf{X}_\mu$. Dans ce cas, on a une *formule de transition* :

$$\mathbf{X}_\nu = x_r \mathbf{X}_\mu + \sum_\eta \mathbf{X}_\eta \quad , \tag{1.36}$$

qui permet une induction positive sur les polynômes de Schubert. L'itération des transitions donne un algorithme très efficace pour développer un polynôme de Schubert $\mathbf{X}_\nu$ dans la base des monômes. Cet algorithme est notamment utilisé dans la fonction `X2x` mais aussi dans la fonction `TableX` dont nous avons parlé précédemment.

Négligeant le terme $x_r \mathbf{X}_\mu$, *i.e.* annulant x_r, on écrit :

$$\mathbf{X}_\nu \longrightarrow \sum_\eta \mathbf{X}_\eta \quad , \tag{1.37}$$

que l'on appelle "transition tronquée".

La *transition maximale* est le cas où l'on prend dans (1.36) comme indice r la position de la dernière descente de ν (pour ν différent de l'identité). La transition maximale est donnée par la fonction `MaxTr` comme le montre l'exemple suivant :

```
ACE> MaxTr([3,1,5,2,4]);

  X[3, 1, 5, 2, 4] = x3 X[3, 1, 4, 2, 5] + X[3, 4, 1, 2, 5] + X[4, 1, 3, 2, 5]
```

On associe alors à chaque permutation ν un arbre défini récursivement comme suit : si ν est vexillaire, l'arbre est réduit à une feuille indicée par ν ; sinon, considérant l'équation de transition maximale tronquée $\mathbf{X}_\nu \longrightarrow \sum \mathbf{X}_\eta$, on construit toutes les branches reliant ν à tous les sous-arbres correspondant à chaque η. La lecture des feuilles η de l'arbre associé à $\mathbf{X}_\nu$ peut s'écrire :

$$\mathbf{X}_\nu \equiv \sum \mathbf{X}_\eta \quad ,$$

c'est-à-dire que modulo l'annulation des x_r rencontrées dans l'algorithme de transition, le polynôme de Schubert est congru à une somme de polynômes vexillaires. On pourrait en fait supposer que les permutations η sont grassmanniennes. Dans ce cas, les $\mathbf{X}_\eta$ sont des fonctions de Schur et la somme $\sum \mathbf{X}_\eta$ est ce qu'on appelle la "partie symétrique" du polynôme de Schubert $\mathbf{X}_\eta$; on peut aussi l'obtenir avec les différences divisées isobares.

Comme cas particulier (en prenant comme permutation de départ un produit direct de deux permutations grassmanniennes) on obtient un algorithme pour le produit de deux fonctions de Schur. Cet algorithme totalement différent de la *formule de Littlewood-Richardson* est présent dans la bibliothèque `SP` au travers de la fonction `LRrbyX`. Par exemple, les lignes suivantes montrent le calcul du développement du produit $s_{3,1}s_{2,1}$ en termes de fonctions de Schur :

```
ACE> LRrbyX(s[3,1], s[2,1]);

        s[5, 2] + s[5, 1, 1] + s[4, 3] + 2 s[4, 2, 1] + s[4, 1, 1, 1]

                + s[3, 3, 1] + s[3, 2, 2] + s[3, 2, 1, 1]
```

La géométrie des variétés de drapeaux fournit un produit scalaire sur $\mathbb{Z}[x_1, \ldots, x_n]$ en tant que module libre sur l'anneau des polynômes symétriques $Sym[x_1, \ldots, x_n]$. Ce produit scalaire est défini à l'aide de l'opérateur ∂_ω, $(\omega \in \mathfrak{S}_n)$:

$$\langle f, g \rangle := \partial_\omega(fg) \quad \in Sym[x_1, \ldots, x_n] \quad . \tag{1.38}$$

Pour ce produit scalaire, les différences divisées ∂_i, $i = 1, \ldots, n-1$ sont auto-adjointes, *i.e.* $\langle \partial_i f, g \rangle = \langle f, \partial_i g \rangle$. En outre, soient $\mu, \nu \in \mathfrak{S}_n$ telles que $l(\mu) + l(\nu) = \binom{n}{2}$ alors les polynômes de Schubert $\mathbf{X}_\mu$ et $\mathbf{X}_\nu$ vérifient la propriété suivante :

$$\langle \mathbf{X}_\mu, \mathbf{X}_\nu \rangle = \begin{cases} 1 & \text{si } \nu = \omega\mu \quad , \\ 0 & \text{autrement} \quad . \end{cases} \tag{1.39}$$

En d'autres termes, les polynômes de Schubert de degré k sont adjoints aux polynômes de Schubert de degré $\binom{n}{2} - k$. Montrons par exemple comment calculer la table des produits scalaires des polynômes de Schubert, à l'aide de la fonction `ScalarPol` :

```
ACE> Sn:=3:
ACE> scal:=array(1..Sn!, 1..Sn!):
ACE> l:=ListPerm(Sn):
ACE> for i from 1 to Sn! do
       for j from i to Sn! do
         scal[i, j]:=ScalarPol(X2x(X[op(op(i, l))]), X2x(X[op(op(j, l))]), Sn);
         scal[j, i]:=scal[i, j]
       od
     od:
ACE> print(l);

      [[1, 2, 3], [1, 3, 2], [3, 1, 2], [2, 1, 3], [2, 3, 1], [3, 2, 1]]

ACE> print(scal);

                [ 0    0     0     0      0        1    ]
                [                                       ]
                [ 0    0     1     0      0      s[1]   ]
                [                                       ]
                [ 0    1     0     0     s[1]    s[2]   ]
                [                                       ]
                [ 0    0     0     0      1      s[1]   ]
                [                                       ]
                [ 0    0    s[1]   1      0     s[1, 1] ]
                [                                       ]
                [ 1  s[1]  s[2]  s[1]  s[1, 1]  s[2, 1] ]
```

La bibliothèque fournit aussi les polynômes de Schubert doubles. Considérons deux ensembles indépendants d'indéterminées $\mathbf{x} := \{x_1, \ldots, x_n\}$ et $\mathbf{y} := \{y_1, \ldots, y_n\}$. Le *polynôme de Schubert double maximal* $\mathbb{X}_\omega$ est défini par :

$$\mathbb{X}_\omega(\mathbf{x}, \mathbf{y}) := \prod_{i+j \leq n} (x_i - y_j) \quad , \tag{1.40}$$

où ω dénote l'élément maximal de $\mathfrak{S}_n$. Plus généralement, pour toute permutation $\mu \in \mathfrak{S}_n$, le *polynôme de Schubert double* $\mathbb{X}_\mu$ est défini comme image par différence divisée de $\mathbb{X}_\omega(\mathbf{x}, \mathbf{y})$:

$$\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) := \partial_{\mu^{-1}\omega}\Big(\mathbb{X}_\omega(\mathbf{x}, \mathbf{y})\Big) \quad , \tag{1.41}$$

les différences divisées n'agissant que sur les variables $\mathbf{x}$.

La fonction `TableXX` calcule la table de tous les polynômes de Schubert doubles pour un groupe symétrique donné :

```
ACE> TableXX(3);
                  table([
                      [1, 2, 3] = 1
                      [2, 1, 3] = x1 - y1
                      [1, 3, 2] = x1 - y1 + x2  - y2
                      [2, 3, 1] = (x1 - y1) (x2 - y1)
                      [3, 1, 2] = (x1 - y1) (x1 - y2)
                      [3, 2, 1] = (x1 - y1) (x1 - y2) (x2 - y1)
                  ])
```

Nous avons déjà vu précédemment la formule d'interpolation de Newton (1.34) dans le cas d'un polynôme en une variable x_1. Les polynômes de Schubert doubles généralisent les coefficients de Newton $(x_1 - y_1) \ldots (x_1 - y_i)$ au cas de plusieurs variables. En d'autres termes, soit $p \in \mathbb{Z}[x_1, x_2, \ldots]$ et $\mathbf{y} := \{y_1, \ldots, y_n\}$ un deuxième ensemble de variables. Alors, la formule d'interpolation de Newton se généralise en :

$$p(\mathbf{x}) \equiv \sum_{\mu \in \mathfrak{S}_n} \partial_\mu(p)_{|\mathbf{x}=\mathbf{y}} \, \mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) \quad . \tag{1.42}$$

Cette formule, due à Lascoux et Schützenberger [50], est une formule d'interpolation. Pour n suffisament grand, on obtient une égalité, c'est-à-dire l'expression exacte du polynôme p dans la base des polynômes de Schubert doubles. Cette expression est aussi donnée par la fonction `x2XX` qui utilise un algorithme entièrement différent.

La commande suivante montre l'utilisation de la fonction `NewtonInterp` qui réalise l'interpolation de Newton, le dernier argument étant le degré du groupe symétrique choisi :

```
ACE> NewtonInterp(x1^2, 2);

                            2
                          y1  XX[1, 2] + (y1 + y2) XX[2, 1]
ACE> XX2x(");

                                y1 x1 + y2 x1 - y2 y1

ACE> Tox( NewtonInterp(x1^5 + x1*x2^3*x3, 6) );

                                     5        3
                                   x1  + x1 x2  x3
```

Dans cet exemple, la fonction `XX2x` convertit n'importe quelle expression mettant en jeu des polynômes de Schubert doubles sur la base des monômes et la fonction `Tox` développe n'importe quelle expression sur la base des monômes.

Puisque dans la définition (1.41) les différences divisées n'agissent que sur les $\mathbf{x}$, les polynômes de Schubert doubles sont des sommes de monômes x^α, $\alpha \subseteq \delta$ à coefficients en $\mathbf{y}$. De fait, ils peuvent s'exprimer comme combinaisons linéaires de polynômes de Schubert simples (en les variables $\mathbf{x}$) à coefficients dans $\mathbb{Z}[\mathbf{y}]$.

Comme pour le cas simple, il existe une formule de Monk pour multiplier un polynôme de Schubert double par une variable x_r [42] et r peut être choisi de telle sorte que l'on ait :

$$\mathbb{X}_\nu = (x_r - y_{\mu(r)})\mathbb{X}_\mu + \sum_\eta \mathbb{X}_\eta \quad . \tag{1.43}$$

Cela donne à nouveau un moyen efficace de calculer le développement d'un polynôme de Schubert double dans la base des monômes. En fait, en algèbre et géométrie on a besoin de spécialiser $\{y_1, \ldots, y_n\}$ en une permutation de $\{x_1, \ldots, x_n\}$ et l'algorithme précédent est compatible avec cette spécialisation. C'est pourquoi la fonction `TableXX` permet une telle spécialisation lorsqu'elle est appelée avec un deuxième paramètre spécifiant cette permutation :

```
ACE> TableXX(4, [2,4,1,3]);

   table([
     [2, 4, 3, 1] = 0
     [2, 1, 3, 4] = x1 - x2
     [2, 3, 4, 1] = 0
     [4, 2, 1, 3] = 0
     [2, 3, 1, 4] = 0
     [3, 4, 1, 2] = 0
     [3, 1, 4, 2] = (x3 - x4) (- x4 + x1) (x1 - x2)
     [1, 2, 3, 4] = 1
     [3, 4, 2, 1] = 0
     [4, 2, 3, 1] = 0
     [3, 2, 4, 1] = 0
     [3, 2, 1, 4] = 0
     [2, 4, 1, 3] = 0
     [4, 1, 2, 3] = 0
     [1, 3, 2, 4] = - x4 + x1
     [1, 4, 2, 3] = (x2 - x1) (- x4 + x1) + (- x4 + x1) (x1 - x2)
     [4, 3, 2, 1] = 0
     [1, 2, 4, 3] = x3 - x4
     [2, 1, 4, 3] = (x3 - x1) (x1 - x2) + (- x4 + x1) (x1 - x2)
     [4, 1, 3, 2] = 0
     [3, 1, 2, 4] = (- x4 + x1) (x1 - x2)
     [1, 4, 3, 2] = (x3 - x4) ((x2 - x1) (- x4 + x1) + (- x4 + x1) (x1 - x2))
     [4, 3, 1, 2] = 0
     [1, 3, 4, 2] = (x3 - x4) (- x4 + x1)
   ])
```

où $y_1 \to x_2$, $y_2 \to x_4$, $y_3 \to x_1$ et $y_4 \to x_3$.

1.15 Les fonctions symétriques comme scalaires

L'un de nos derniers développements concerne l'exploration d'un point de vue particulier pour calculer dans l'anneau des polynômes à plusieurs variables. Ce point de vue s'appuie sur plusieurs résultats que nous avons déjà rencontrés et la bibliothèque `FM` utilise plusieurs des bibliothèques dont nous avons parlé précédemment.

Les calculs dans l'anneau $\mathbb{Z}[x_1, \dots, x_n]$ deviennent vite explosifs, car la taille des objets manipulés croît très rapidement en fonction du nombre de variables et du degré des polynômes. L'un des problèmes consiste à trouver des bases linéaires bien adaptées à certains problèmes sans perdre pour autant la trace de la structure multiplicative. Dans le cas d'une variable, de telles bases sont fournies par la théorie de l'interpolation ou celle des polynômes orthogonaux. Dans le cas de plusieurs variables, on cherche aussi des bases orthogonales, mais l'on veut aussi prendre en compte les symétries partielles ce que ne permet pas la base usuelle des monômes.

Comme nous l'avons vu, lorsque les polynômes sont totalement symétriques, on dispose des techniques combinatoires puissantes de la théorie des fonctions symétriques pour prendre en compte des calculs algébriques. Pour étendre ces méthodes au cas de polynômes non symétriques, on considère l'anneau des polynômes en n variables comme module sur l'anneau $Sym[x_1, \dots, x_n]$ des polynômes symétriques. Il est facile de montrer que l'ensemble des monômes $x_1^{i_1} \dots x_n^{i_n}$, $(i_1, \dots, i_n) \leq (n-1, \dots, 0)$ composante à composante forme une base linéaire de ce module, qui est libre et de rang fini $n!$.

Nous utiliserons cependant une base plus intéressante, celle des polynômes de Schubert simples $\{\mathbf{X}_\mu, \mu \in \mathfrak{S}_n\}$. Au lieu de décomposer le polynôme $p := x_2^2x_1x_3^3 + x_2x_1^2x_3^3 + 2x_2^2x_1^2x_3^2 + x_2^3x_1x_3^2 + x_2x_1^3x_3^2 + 2x_2^3x_1^2x_3 + 2x_2^2x_1^3x_3 + x_2^4x_1x_3 + x_2x_1^4x_3 + x_2^4x_1^2 + x_2^3x_1^3 + x_2^2x_1^4 + x_3^2x_2^3 + 2x_3^2x_2^2x_1 + 2x_3^2x_2x_1^2 + x_3^2x_1^3 + x_3x_2^3x_1 + 2x_3x_2^2x_1^2 + x_3x_2x_1^3 + x_2^3x_1^2 + x_2^2x_1^3$ sur la base des polynômes de Schubert indicés par des permutations de $\mathfrak{S}_\infty$: $p = \mathbf{X}_{2,4,6,1,3,5} + \mathbf{X}_{1,5,4,2,3} + \mathbf{X}_{2,4,5,1,3} + \mathbf{X}_{2,6,3,1,4,5} + \mathbf{X}_{3,6,1,2,4,5}$, on peut utiliser comme coefficients, des fonctions symétriques en x_1, x_2, x_3 : $p = (x_3^3x_2 + x_3^3x_1 + 2x_3^2x_2x_1 + x_3^2x_2^2 + x_3^2x_1^2 + 2x_3x_2^2x_1 + 2x_3x_2x_1^2 + x_3x_2^3 + x_3x_1^3 + x_2^3x_1 + x_2^2x_1^2 + x_2x_1^3)\mathbf{X}_{2,3,1} + (x_3^2x_2^2 + x_3^2x_2x_1 + x_3^2x_1^2 + x_3x_2^2x_1 + x_3x_2x_1^2 + x_2^2x_1^2)\mathbf{X}_{1,3,2}$. Si l'on exprime ces coefficients sur la base des fonctions de Schur, on obtient une expression très compacte de p :

$$p \;=\; s_{3,1}\mathbf{X}_{2,3,1} + s_{2,2}\mathbf{X}_{1,3,2} \quad .$$

Remarque. Cette expression révèle que p est symétrique en x_1, x_2 car elle n'implique que des polynômes de Schubert ayant cette même propriété de symétrie (polynômes de Schubert indicés par des permutations croissantes sur les deux premières places).

A partir de maintenant, nous fixons l'entier n et nous utilisons uniquement les polynômes de Schubert $\mathbf{X}_w, w \in \mathfrak{S}_n$. Soit $Sym[\mathbf{x}] := Sym[x_1, \dots, x_n]$ le sous-anneau des polynômes symétriques en $x_1, \dots, x_n$. L'espace $\mathbb{Z}[\mathbf{x}]$ est un module libre de dimension $n!$ sur l'anneau $Sym[\mathbf{x}]$ avec comme base les polynômes de Schubert simples $\mathbf{X}_w, w \in \mathfrak{S}_n$:

$$\mathbb{Z}[x_1, \dots, x_n] \quad = \quad \bigoplus_{w \in \mathfrak{S}_n} Sym[x_1, \dots, x_n]\; \mathbf{X}_w \quad . \tag{1.44}$$

En d'autres termes, tout polynôme $p \in \mathbb{Z}[\mathbf{x}]$ peut s'exprimer de manière unique comme une somme :

$$p = \sum_{w \in \mathfrak{S}_n} c_w \, \mathbf{X}_w \quad ,$$

où les c_w appartiennent à $Sym[\mathbf{x}]$.

Comme nous l'avons déjà expliqué, la formule de Monk donne la structure multiplicative de l'anneau des polynômes en une infinité de variables mais elle ne préserve pas le sous-espace ayant comme base les $\mathbf{X}_w, w \in \mathfrak{S}_n$. Puisque dans le produit $x_r\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$, il apparaît au plus un terme indicé par une permutation dans $\mathfrak{S}_{n+1}\backslash\mathfrak{S}_n$, nous sommes naturellement conduit à chercher à exprimer certains polynômes de Schubert simples comme somme à coefficients dans $Sym[x_1, \ldots, x_n]$ de polynômes de Schubert indicés par des permutations de $\mathfrak{S}_n$. Plus généralement, tout polynôme de Schubert de $\mathbb{Z}[x_1, \ldots, x_n]$ a une expression dans la base Schubert $\mathbf{X}_w, w \in \mathfrak{S}_n$.

Les calculs dans l'anneau $\mathbb{Z}[x_1, \ldots, x_n]$ en tant que module libre sur $Sym[x_1, \ldots, x_n]$ de base les $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$ mettent en œuvre la combinatoire des fonctions symétriques, celle des permutations et celle des polynômes de Schubert, mais nécessitent aussi de nouveaux algorithmes. Nous en avons fait une bibliothèque `FM` (*Free Module*).

Cette bibliothèque fournit les fonctions de base pour calculer avec cette structure de module sur l'anneau des polynômes symétriques, avec comme base des fonctions symétriques la base des fonctions de Schur. Nous avons modifié la formule de Monk (1.35) en une formule n'impliquant que des permutations de $\mathfrak{S}_n$. La fonction `Monkfix` permet ainsi d'exprimer le produit d'un polynôme de Schubert $\mathbf{X}_w, w \in \mathfrak{S}_n$ par une variable $x_i, i = 1, \ldots, n$, fonction à partir de laquelle on peut plus généralement exprimer le produit d'un polynôme de Schubert $\mathbf{X}_w, w \in \mathfrak{S}_n$ par n'importe quel polynôme en $x_1, \ldots, x_n$. La fonction `FM_n` est utilisée pour fixer le nombre de variables. L'exemple suivant permet de comparer les résultats des fonctions `Monk` (de la bibliothèque `SP`) et `Monkfix`:

```
ACE> Monk(3, X[3,1,4,2]);

              - X[3, 4, 1, 2] - X[4, 1, 3, 2] + X[3, 1, 5, 2, 4]

ACE> FM_n(4);

                                      4

ACE> Monkfix(3, X[3,1,4,2]);

       - X[3, 4, 1, 2] - X[4, 1, 3, 2] + s[1, 1, 1, 1] X[1, 2, 3, 4]

            - s[1, 1, 1] X[2, 1, 3, 4] + s[1] X[3, 1, 4, 2]
```

Plus généralement, la bibliothèque permet de multiplier n'importe quelle combinaison linéaire de polynômes de Schubert à coefficients dans $Sym\,[x_1, \ldots, x_n]$ par un polynôme à coefficients aussi dans $Sym\,[x_1, \ldots, x_n]$. On met donc en œuvre des algorithmes spécifiques aux fonctions symétriques, ces algorithmes étant basés sur des propriétés combinatoires des partitions et autres tableaux de Young, lesquelles sont indépendantes du nombre de variables. En ce sens, ce point de vue sur l'anneau des polynômes présente l'intérêt d'intégrer toutes les techniques déjà disponibles sur les fonctions symétriques dans des calculs de polynômes généraux ou présentant des symétries partielles.

La fonction `ToXfix` est utile pour convertir n'importe quelle expression dans la base Schubert indicée par les permutations de $\mathfrak{S}_n$:

```
ACE> FM_n(4);

                                      4

ACE> ToXfix(x3^3*X[2,4,1,3]^2, collect);

     s[2, 1, 1, 1] X[3, 4, 1, 2] - s[2, 2, 1, 1] X[1, 4, 3, 2]

    + (s[2, 2, 1] + s[2, 1, 1, 1]) X[2, 4, 3, 1] + s[2, 2, 2, 2] X[1, 3, 2, 4]

    - s[2, 1, 1] X[3, 4, 2, 1] + (s[2, 2, 2] - s[2, 2, 1, 1]) X[2, 3, 4, 1]

    + s[2, 2, 2, 2] X[1, 2, 4, 3] - s[2, 2, 2, 1] X[1, 3, 4, 2]
```

Nous fournissons aussi une fonction `ToXfixScal` qui donne exactement le même résultat que la fonction `ToXfix` mais en utilisant des produits scalaires plutôt que la formule de Monk modifiée.

Nous verrons un peu plus loin quelques applications de cette bibliothèque qui reflète d'ailleurs l'enjeu de la mise en place d'un environnement de calcul cohérent. Le développement de `FM` aurait été bien plus long dans un contexte où il aurait fallu programmer tous les algorithmes impliquant le groupe symétrique, les fonctions symétriques, les différences divisées ou les polynômes de Schubert. En outre, `FM` met en œuvre des résultats mathématiques très récents et permet l'expérimentation dans des domaines en plein essor.

En résumé, l'environnement `ACE` tente de couvrir plusieurs domaines spécialisés de la combinatoire algébrique au sein du système généraliste MAPLE. Les quelques pages de présentation qui précèdent montrent combien ces différentes théories sont étroitement liées. Tout un éventail d'outils[(4)] au service d'un unique enjeu : expérimenter en calculant.

Un dernier mot sur la bibliothèque `ACE`, qui inclut toutes les autres et propose quelques fonctions d'utilité générale. Par exemple, puisque la bibliothèque `FM` adopte le point de vue d'une base finie pour l'anneau des polynômes, nous pouvons pré-calculer les constantes de structures correspondant à la base Schubert et stocker ces résultats dans des tables. Les algorithmes de calcul contrôlent si les tables sont chargées et dans ce cas n'effectuent aucun calcul, allant simplement chercher les résultats dans ces tables. L'environnement est ainsi distribué avec quelques tables et la bibliothèque `ACE` accueille les fonctions permettant de charger (`LoadData`), décharger (`UnloadData`), construire et sauver des tables (`BuildData`).

[(4)] Nous n'avons présenté que 15 bibliothèques parmi les 16. En fait, celle que nous avons volontairement oubliée est très proche de la bibliothèque `SYMF`, contenant essentiellement les mêmes fonctionnalités mais permettant de manipuler les caractères des représentations du groupe linéaire $GL(n)$. La bibliothèque `CLG` reprend ainsi un calcul de fonctions symétriques qui codent ces caractères mais cette fois sur un alphabet fixe ce qui conduit à des formules tronquées. Par ailleurs, nous ne détaillons pas les bibliothèques de gestion des erreurs, d'analyse des types ou encore de sorties de résultats au format TEX.

Références bibliographiques

[1] S. Ariki. On the decomposition numbers of the Hecke algebra of $G(m,1,n)$, (1996). preprint.

[2] F. Bergeron and D. Krob. Acyclic complexes related to noncommutative symmetric functions, (1995). preprint, to appear in J. Alg. Comb.

[3] G.R.E. Black, R.C. King, and B.G. Wybourne. Kronecker products for compact semisimple Lie groups. *J. Phys. A: Math. Gen.*, (1983).

[4] R.W. Carter. Representation theory of the 0-Hecke algebra. *J. of Algebra*, 15: 89–103, (1986).

[5] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *First Leaves: A Tutorial Introduction To Maple V.* Springer-Verlag, (1992).

[6] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, (1992).

[7] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *Maple V Library Reference Manual.* Springer-Verlag, (1992).

[8] I.V. Cherednik. A new interpretation of Gelfand-Tzetlin bases. *Duke. Math. J.*, 54: 563–577, (1987).

[9] C.W. Curtis and I. Reiner. *Methods of representation theory with applications to finite groups and orders*, volume 1. Wiley, (1990).

[10] M. Demazure. Désingularisation des variétés de Schubert. *Annales E.N.S.*, 6: 163–172, (1974).

[11] R. Dipper and G. James. Blocks and idempotents of Hecke algebras of general linear groups. In *Proc. London Math. Soc.*, volume 54, pages 57–82, (1987).

[12] G. Duchamp, D. Krob, B. Leclerc, and J.-Y. Thibon. Fonctions quasi-symétriques, fonctions symétriques non commutatives, et algèbres de Hecke à $q = 0$. *Comptes Rendus de l'Académie de Sciences*, 322: 107–112, (1996).

[13] S. Fomin and C. Greene. Noncommutative Schur functions and their applications, (1995). preprint.

[14] S. Fomin and A. Kirillov. Yang-Baxter equation, symmetric functions and Schubert polynomials. In *Proceedings of the 5-th Conference "Formal Power Series and Algebraic Combinatorics"*, pages 215–229, Florence, (1993).

[15] S. Fomin and A.N. Kirillov. Combinatorial B_n-analogues of Schubert polynomials, (1994). preprint.

[16] W. Fulton. Flags, Schubert polynomials, degeneracy loci and determinantal formulas. *Duke Math. J.*, 65: 381–420, (1991).

[17] W. Fulton and J. Harris. *Representation theory, a first course.* Springer-Verlag, (1991).

[18] A.M. Garsia and C. Reutenauer. A decomposition of Solomon's descent algebra. *Adv. in Math.*, 77: 189–262, (1989).

[19] I.M. Gelfand, D. Krob, A. Lascoux, B. Leclerc, V.S. Retakh, and J.-Y. Thibon. Noncommutative symmetric functions. *Advances in Mathematics*, 112: 218–348, (1995).

[20] I.M. Gelfand and V.S. Retakh. Determinants of matrices over noncommutative rings. *Funct. Anal. Appl.*, 25: 91–102, (1991).

[21] I.M. Gelfand and V.S. Retakh. A theory of noncommutative determinants and characteristic functions of graphs. *Funct. Anal. Appl.*, 26: 1–20, (1992). Publ. LACIM, UQAM, Montréal, **14**, 1–26.

[22] H. Georgi. *Lie algebras in particle physics.* Benjamin, Reading Mass., (1982).

[23] I. Gessel. Multipartite P-partitions and inner product of skew Schur functions. *Contemporary Math.*, 34: 289–301, (1984).

[24] I. Grojnowski. Affine Hecke algebras (and affine quantum GL_n) at roots of unity. *Internat. Res. Math. Notices*, 5: 215–217, (1994).

[25] T. Hayashi. q-analogues of Clifford and Weyl algebras – spinor and oscillator representations of quantum enveloping algebras. *Commun. Math. Phys.*, 127: 129–144, (1990).

[26] F. Hirzebruch. *Topological methods in algebraic geometry.* Springer-Verlag, (1978). 3rd ed.

[27] J.E. Humphreys. *Reflection groups and Coxeter groups.* Cambridge University Press, Springer-Verlag, (1990).

[28] N. I.Bernstein, M. I.Gelfand, and S.I. Gelfand. Schubert cells and cohomology of the spaces G/P. *Uspekhi.Mat.Nauk*, 28: 3–26, (1973).

[29] G. James. The decomposition matrices of $GL_n(q)$ for $n \leq 10$. *Proc. London Math. Soc.*, 60: 225–265, (1990).

[30] G. James and A. Kerber. *The representation theory of the symmetric group.* Addison-Wesley, (1981).

[31] M. Kashiwara. On crystal bases of the q-analogue of universal enveloping algebras. *Duke Math. J.*, 63: 465–516, (1991).

[32] M. Kashiwara. Crystal bases of modified quantized enveloping algebra. *Duke Math. J.*, 73: 383–413, (1994).

[33] M. Kashiwara, T. Miwa, and E. Stern. Decomposition of q-deformed Fock spaces, (1995). preprint RIMS-1035, Selecta Mathematica (New series), (1996).

[34] A. Kerber. *Algebraic combinatorics via finite group actions.* Mannheim: BI-Wissenschaftsverlag, (1991).

[35] R.C. King. Modification rules and products of irreducible representations of the unitary, orthogonal and symplectic groups. *J. Math. Phys.*, 12: 1588–1598, (1971).

[36] R.C. King. *S-functions and characters of Lie algebras and Lie groups*, volume 26 of *Topics in Algebra.* Banach Center Publication, PWN, Warsaw, (1990).

[37] A.S. Kleshchev. Branching rules for the modular representations of symmetric groups III; some corollaries and a problem of Mullineux. *J. London Math. Soc.*, 2, (1995).

[38] A. Klyachko. Lie elements in the tensor algebra. *Siberian Math. J.*, 15: 1296–1304, (1974).

[39] D.E. Knuth. Permutations, matrices and generalized Young tableaux. *Pacific J. Math.*, 34: 709–727, (1970).

[40] D. Knutson. λ-rings and the representation theory of the symmetric group. *Springer L.N.M.*, 308, (1973).

[41] A. Kohnert. Weintrauben, Polynome, Tableaux (thèse). *Bayreuther Math. Schr.*, 38: 1–97, (1991).

[42] A. Kohnert and S. Veigneau. Using Schubert toolkit to compute with polynomials in several variables. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 283–293, Minneapolis, (1996). University of Minnesota.

[43] K. Koike and I. Terada. Young-diagrammatic methods for the representation theory of the classical groups of type B_n, C_n, D_n. *J. Algebra*, 107: 466–511, (1987).

[44] C. Kostka. Über den Zusammenhang zwischen einingen Formen von symmetrischen Funktionen.

[45] D. Krob, A. Lascoux, B. Leclerc, J.-Y. Thibon, B.C.V. Ung, and S. Veigneau. Algebraic Combinatorics with MAPLE and ACE. *Maple Technical Newsletter*, (1996). à paraître.

[46] A. Lascoux, B. Leclerc, and J.-Y. Thibon. Fonctions de Hall-Littlewood et polynômes de Kostka-Foulkes aux racines de l'unité. *Comptes Rendus de l'Académie des Sciences de Paris*, 316: 1–6, (1993).

[47] A. Lascoux, B. Leclerc, and J.-Y. Thibon. Flag varieties and the Yang-Baxter equation. *Letters in Mathematical Physics*, (1996). soumis.

[48] A. Lascoux, B. Leclerc, and J.-Y. Thibon. Hecke algebras at roots of unity and crystal bases of quantum affine algebras. *Communications in Mathematical Physics*, (1996). à paraître.

[49] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

[50] A. Lascoux and M.P. Schützenberger. Interpolation de Newton à plusieurs variables. *Springer L.N.*, 1146: 161–175, (1985).

[51] A. Lascoux and M.P. Schützenberger. Schubert polynomials and the Littlewood-Richardson rule. *Lett. Math. Phys.*, 10: 111–124, (1985).

[52] D.E. Littlewood. *The theory of group characters.* Oxford, (1950). 2nd ed.

[53] D.E. Littlewood. Products and plethysms of characters with orthogonal, symplectic and symmetric groups. *J. Can. Math.*, 10: 17–32, (1958).

[54] D.E. Littlewood. On certain symmetric functions. *J. London. Math. Soc.*, 11: 485–498, (1961).

[55] D.E. Littlewood and A.R. Richardson. Group characters and algebra.

[56] G. Lusztig. Singularities, character formulas and a q-analog of weight multiplicities. *Analyse et topologie des espaces singuliers (II-III), Astérisque*, 101–102, (1983).

[57] I.G. Macdonald. *Notes on Schubert polynomials*, volume 6. Publ. LACIM, Montréal, (1991).

[58] I.G. Macdonald. *Symmetric functions and Hall polynomials.* Oxford: Clarendon Press, (1995). 2nd ed.

[59] P.A. MacMahon. *Combinatory analysis.* Cambridge University Press, (1915). Chelsea reprint, 1960.

[60] C. Malvenuto. *Produits et coproduits des fonctions quasi-symmétriques et de l'algèbre des descentes (Thèse).* Publ. LACIM, UQAM, Montréal, (1993).

[61] C. Malvenuto and C. Reutenauer. Duality between quasi-symmetric functions and the solomon descent algebra, (1993). preprint UQAM.

[62] B. Mielnik and J. Plebański. Combinatorial approach to Baker-Campbell-Hausdorff exponents. *Ann. Inst. Henri Poincaré*, XII: 215–254, (1970).

[63] K.C. Misra and T. Miwa. Crystal base of the basic representation of $U_q(\widehat{sl}_n)$. *Commun. Math. Phys.*, 134: 79–88, (1990).

[64] D. Monk. The Geometry of flag manifolds. *Proc. London M.S. (3)*, 9: 253–286, (1959).

[65] R.V. Moody and J. Patera. Fast recursion formula for weight multiplicities.

[66] P. Pragacz and J. Ratajski. Formulas for Lagrangian and orthogonal degeneraci loci; the $\widetilde{Q}$-Polynomials Approach. to appear in Compositio Math.

[67] D. Redfern. *The Maple handbook, Maple V Release 3.* Springer-Verlag, (1993).

[68] C. Reutenauer. *Free Lie algebras.* Oxford University Press, (1993).

[69] D.H. Sattinger and O.L. Weaver. *Lie groups and algebras with applications to physics, geometry and mechanics*, volume 61. Springer-Verlag, New York, (1986).

[70] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13: 179–191, (1961).

[71] M.P. Schützenberger. Propriétés nouvelles des tableaux de Young. In *Séminaire Delange-Pisot-Poitou*, volume 26, (1977).

[72] J.-P. Serre. *Complex semisimple Lie algebras.* Springer-Verlag, New York, (1987).

[73] L. Solomon. A Mackey formula in the group ring of a Coxeter group. *J. of Algebra*, 41: 255–268, (1976).

[74] D. Stanton. *Invariant theory and tableaux.* Springer-Verlag, New York, (1986).

[75] J. Stembridge. *SF, a maple package for symmetric functions.* University of Michigan, (1993).

[76] B.C.V. Ung and S. Veigneau. ACE, un environnement de calcul en combinatoire algébrique. In *Proc. 7-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 557–562, Noisy-le-Grand, (1995). Université de Marne-la-Vallée.

[77] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 1.0. Rapport 95–14, IGM, (1995).

[78] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 2.0. Rapport, IGM, (1996).

[79] S. Veigneau. SP, a package for Schubert polynomials realized with the computer algebra system Maple. *Journal of Symbolic Computation*, (1996). à paraître.

[80] H. Weyl. *Classical groups.* Princeton University Press, Princeton, NJ, (1939).

[81] B.G. Wybourne. *SCHUR an interactive program for calculating properties of Lie groups.*

[82] B.G. Wybourne. *Symmetry principles and atomic spectroscopy.* Wiley, (1969).

2

Exemples combinatoires

Aucune règle n'existe,
les exemples ne viennent qu'au secours des règles en peine d'exister.

— ***André Breton***, *Le Surréalisme et la Peinture.*

Dans le chapitre précédent, nous avons parcouru les bibliothèques qui constituent la deuxième version de l'environnement ACE. Nous nous intéressons dans ce chapitre à l'utilisation de l'environnement en présentant plusieurs exemples combinatoires. Nous voyons tout d'abord quelques exemples sur les tableaux de Young et les fonctions symétriques, l'interprétation combinatoire des fonctions de Schur comme sommes de tableaux de Young et la règle de Littlewood-Richardson. Nous donnons ensuite une application des fonctions symétriques à des calculs sur les caractères du groupe symétrique (opérateurs d'Adams). Nous continuons avec le calcul des fonctions génératrices des polynômes de Schubert associés au groupe symétrique et au groupe hyperoctaédral et nous terminons par le développement du q-discriminant.

2.1 Tableaux de Young et fonctions symétriques

L'ensemble des tableaux de Young de forme $(3,2,1)$ et de poids $(2,2,1,1)$ peut être engendré par la commande suivante :

```
ACE> LT1:=ListTab([3,2,1], [2,2,1,1]);

          LT1 := [[[4], [2, 3], [1, 1, 2]], [[4], [2, 2], [1, 1, 3]],

                   [[3], [2, 4], [1, 1, 2]], [[3], [2, 2], [1, 1, 4]]]
```

Les tableaux sont représentés par des listes de lignes ou par des mots. L'environnement permet d'afficher les tableaux comme des objets en deux dimensions grâce à la commande Tab2Mat de sorte que l'on obtient pour le calcul précédent la liste :

```
ACE> map(Tab2Mat, LT1);

               [ 4       ]  [ 4       ]  [ 3       ]  [ 3       ]
               [         ]  [         ]  [         ]  [         ]
              [[ 2  3    ], [ 2  2    ], [ 2  4    ], [ 2  2    ]]
               [         ]  [         ]  [         ]  [         ]
               [ 1  1  2 ]  [ 1  1  3 ]  [ 1  1  2 ]  [ 1  1  4 ]
```

Lascoux et Schützenberger ont défini sur les tableaux une statistique appelée charge ([11], [7]), laquelle fournit une interprétation combinatoire des *polynômes de Kostka-Foulkes*. On peut calculer la charge d'un tableau sur sa représentation comme mot :

```
ACE> L:=map(Tab2Word, LT1);

              L := [[4, 2, 3, 1, 1, 2], [4, 2, 2, 1, 1, 3],

                    [3, 2, 4, 1, 1, 2], [3, 2, 2, 1, 1, 4]]

ACE> A:=map(Word2Charge, L);

                             A := [3, 2, 2, 1]
```

ce qui signifie que parmi les tableaux de forme $(3, 2, 1)$ et de poids $(2, 2, 1, 1)$, un tableau a une charge égale à 1, deux tableaux ont une charge égale à 2 et un tableau à 3. Le polynôme générateur de cette statistique est le polynôme de Kostka $K_{321,2211}(q)$, donné par :

```
ACE> K:=convert(map(k->q^k, A), `+`);

                                   3       2
                             K := q  + 2 q  + q
```

Il existe une bijection due à Schensted [10] entre les mots et les couples de tableaux. On note $(P(w), Q(w))$ le couple de tableaux associé au mot w. $P(w)$ et $Q(w)$ sont respectivement appelés P-symbole et Q-symbole de w. L'exemple suivant illustre cette correspondance réalisée par la fonction `Schensted` :

```
ACE> w:=[3,2,1,4,2,5,3,1];

                        w := [3, 2, 1, 4, 2, 5, 3, 1]

ACE> map(Tab2Mat,Schensted(w));
```

```
                    [ 3  4    ]  [ 3  8    ]
                    [         ]  [         ]
                   [[ 2  2  5 ], [ 2  5  7 ]]
                    [         ]  [         ]
                    [ 1  1  3 ]  [ 1  4  6 ]
```

Lascoux et Schützenberger ont introduit [5] une action du groupe symétrique sur les mots qui change l'évaluation d'un mot sans altérer son Q-symbole. La commande suivante permet de vérifier que le Q-symbole du mot `v` obtenu par action de la permutation $[5, 4, 1, 3, 2]$ sur le mot `w` est inchangé :

```
ACE> v:=PermOnWord([5,4,1,3,2], w);

                  v := [4, 2, 1, 5, 3, 5, 4, 1]

ACE> map(Tab2Mat,Schensted(v));

                    [ 4  5    ]  [ 3  8    ]
                    [         ]  [         ]
                   [[ 2  3  5 ], [ 2  5  7 ]]
                    [         ]  [         ]
                    [ 1  1  4 ]  [ 1  4  6 ]
```

Par ailleurs, la charge est aussi invariante sous cette action :

```
ACE> M:=map(w->PermOnWord([2,4,1,3], w), L);

          M := [[4, 3, 4, 1, 2, 2], [4, 2, 3, 1, 2, 4],

               [4, 2, 4, 1, 2, 3], [3, 2, 4, 1, 2, 4]]

ACE> B:=map(Word2Charge,M);

                        B := [3, 2, 2, 1]
```

Les instructions suivantes permettent de comparer les tableaux obtenus aux tableaux originaux :

```
ACE> LT2:=map(Word2Tab, M):
ACE> map(Tab2Mat, LT2);

          [ 4       ]  [ 4       ]  [ 4       ]  [ 3       ]
          [         ]  [         ]  [         ]  [         ]
         [[ 3  4    ], [ 2  3    ], [ 2  4    ], [ 2  4    ]]
          [         ]  [         ]  [         ]  [         ]
          [ 1  2  2 ]  [ 1  2  4 ]  [ 1  2  3 ]  [ 1  2  4 ]
```

L'ensemble des mots ayant un même P-symbole forme une *classe plaxique*. Le contre-tableau est un des représentants distingués d'une classe plaxique. Il est codé en MAPLE par la liste de ses colonnes comme le montre l'exemple suivant :

```
ACE> u:=[5, 2, 4, 1, 6, 3];

                        u := [5, 2, 4, 1, 6, 3]

ACE> Word2CTab(u);

                        [[2], [5, 1], [6, 4, 3]]

ACE> CTab2Mat(");

                              [ 2  5  6 ]
                              [         ]
                              [    1  4 ]
                              [         ]
                              [       3 ]
```

On peut passer du tableau représentant la classe d'un mot u à son contre-tableau par une suite de mouvements dont les règles sont données par le "jeu de taquin" de Schützenberger. Ces mouvements échangent les longueurs de deux colonnes consécutives et les colonnes de gauche des différents représentants de la classe ainsi obtenus sont ordonnées par inclusion. Le contre-tableau obtenu par concaténation de ces colonnes est appelé *clé gauche* de u :

```
ACE> Word2Tab(u);

                        [[5], [2, 4], [1, 3, 6]]

ACE> PermOnTab([2,1,3], ");

                        [[2, 5], [4], [1, 3, 6]]

ACE> PermOnTab([1,3,2], ");

                        [[2, 5], [1, 4, 6], [3]]

ACE> PermOnTab([2,1,3], ");

                        [[2, 5, 6], [1, 4], [3]]
```

Cependant, la clé gauche peut être calculée directement :

```
ACE> Word2LeftKey(u):
ACE> CTab2Mat(");

                               [ 2  2  5 ]
                               [         ]
                               [    1  2 ]
                               [         ]
                               [       1 ]
```

2.2 Interprétation combinatoire des fonctions de Schur

Les fonctions de Schur peuvent être vues comme des sommes de tableaux d'une forme donnée, les entrées i des tableaux de Young étant interprétées comme les variables x_i. La fonction **SchurTab** retourne la liste de tous les tableaux d'une forme donnée sur un alphabet $1, \ldots,$`card` :

```
ListEval:=proc(card, weight)
  local
    i, compo;
  [seq([seq(compo[i]-1, i=1..card)], compo=ListCompo(weight+card, card))]
end;

SchurTab:=proc(part, card)
  local
    compo;
  [seq(op(ListTab(part, compo)), compo=ListEval(card, convert(part, `+`)))]
end;
```

L'exemple suivant montre la liste des tableaux de Young intervenant dans le développement de la fonction de Schur $s_{3,2,2,1}(x_1, x_2, x_3, x_4)$:

```
ACE> map(Tab2Mat, SchurTab([3,2,2,1], 4));

      [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]
      [         ]  [         ]  [         ]  [         ]  [         ]
      [ 3  3    ]  [ 3  4    ]  [ 3  4    ]  [ 3  3    ]  [ 3  4    ]
     [[         ], [         ], [         ], [         ], [         ],
      [ 2  2    ]  [ 2  2    ]  [ 2  3    ]  [ 2  2    ]  [ 2  2    ]
      [         ]  [         ]  [         ]  [         ]  [         ]
      [ 1  1  1 ]  [ 1  1  1 ]  [ 1  1  1 ]  [ 1  1  2 ]  [ 1  1  2 ]
```

```
        [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]
        [         ]  [         ]  [         ]  [         ]  [         ]
        [ 3  3    ]  [ 3  4    ]  [ 3  4    ]  [ 3  3    ]  [ 3  4    ]
        [         ], [         ], [         ], [         ], [         ],
        [ 2  2    ]  [ 2  3    ]  [ 2  2    ]  [ 2  2    ]  [ 2  2    ]
        [         ]  [         ]  [         ]  [         ]  [         ]
        [ 1  1  3 ]  [ 1  1  2 ]  [ 1  1  3 ]  [ 1  1  4 ]  [ 1  1  4 ]

        [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]  [ 4       ]
        [         ]  [         ]  [         ]  [         ]  [         ]
        [ 3  4    ]  [ 3  4    ]  [ 3  4    ]  [ 3  4    ]  [ 3  4    ]
        [         ], [         ], [         ], [         ], [         ]]
        [ 2  3    ]  [ 2  3    ]  [ 2  3    ]  [ 2  3    ]  [ 2  3    ]
        [         ]  [         ]  [         ]  [         ]  [         ]
        [ 1  1  3 ]  [ 1  1  4 ]  [ 1  2  2 ]  [ 1  2  3 ]  [ 1  2  4 ]
```

Enfin, la fonction `Schurx` calcule l'expression d'une fonction de Schur en $x_1, \ldots, x_{\texttt{card}}$ dans la base des monômes :

```
Schurx:=proc(part, card)
  local
    i;
  convert(map(proc(tab, sx)
               convert(subs(sx, Tab2Word(tab)), `*`)
             end, SchurTab(part, card), {seq(i=x.i, i=1..card)}), `+`)
end;
```

Nous continuons l'exemple précédent en donnant le développement de la fonction de Schur $s_{3,2,2,1}$ sur la base des monômes :

```
ACE> Schurx([3,2,2,1], 4);

            2   2   3     2       2   3     2   2      3        2   3   2
        x4 x3  x2  x1  + x4  x3 x2  x1  + x4  x3  x2 x1  + x4 x3  x2  x1

         2      3   2        3   2   2       2   2   2   2     3      2   2
      + x4  x3 x2  x1  + x4 x3  x2  x1  + 3 x4  x3  x2  x1  + x4  x3 x2  x1

         2   3      2     3   2       2     2   2   3        2   3   2
      + x4  x3  x2 x1  + x4  x3  x2 x1  + x4  x3  x2  x1 + x4  x3  x2  x1

         3   2   2
      + x4  x3  x2  x1
```

D'un autre côté, l'environnement nous permet de vérifier que le développement est correct. Cette vérification passe par la base des fonctions monomiales :

```
ACE> CLG_n(4):
ACE> x2m_n( Schurx([3,2,2,1], 4) );

                              m[3, 2, 2, 1] + 3 m[2, 2, 2, 2]

ACE> Tos_n(");

                                     s[3, 2, 2, 1]
```

Notons tout de même que l'environnement fournit un moyen efficace (par les polynômes de Schubert) de développer une fonction de Schur (et plus généralement n'importe quelle fonction symétrique) sur la base des monômes. L'exemple suivant illustre cette possibilité en commençant par fixer la taille de l'alphabet et en affichant uniquement le nombre de monômes de chacun des deux développements envisagés :

```
ACE> CLG_n(4):
ACE> Tox_n(s[5,1,1]): nops(");

                                              80

ACE> Tox_n(s[4,2,1,1]): nops(");

                                              31
```

2.3 La règle de Littlewood-Richardson

La base fondamentale des fonctions symétriques est la base des fonctions de Schur. Nous venons de voir le lien qui existe entre les tableaux de Young d'une forme donnée et les fonctions de Schur mais cela va au-delà puisque la multiplication de deux fonctions de Schur peut être décrite combinatoirement par des manipulations de tableaux. Nous donnons un exemple de cette formule connue sous le nom de "règle de Littlewood-Richardson", exemple dans lequel nous multiplions les fonctions de Schur $s_{5,1,1}$ et $s_{4,2,1,1}$:

```
ACE> LRrSf(s[5,1,1], s[4,2,1,1]);

   3 s[6, 4, 2, 1, 1, 1] + 2 s[8, 3, 2, 2] +  s[8, 4, 1, 1, 1] +  s[8, 3, 3, 1]

 + 3 s[8, 3, 2, 1, 1]    +    2 s[8, 3, 1, 1, 1, 1]    +     2 s[8, 2, 2, 2, 1]

 + 2 s[8, 2, 2, 1, 1, 1]  +  s[7, 5, 2, 1] + s[7, 5, 1, 1, 1]  +  s[7, 4, 3, 1]

 + 3 s[7, 4, 2, 1, 1] + 2 s[7, 3, 3, 1, 1] + 2 s[7, 4, 2, 2] + s[9, 3, 1, 1, 1]
```

```
+ 2 s[7, 4, 1, 1, 1, 1]   +   3 s[7, 3, 2, 2, 1]    +    3 s[7, 3, 2, 1, 1, 1]

+ s[6, 5, 3, 1]   +    2 s[6, 5, 2, 1, 1] + 2 s[6, 4, 3, 1, 1] + s[6, 5, 2, 2]

+ s[6, 5, 1, 1, 1, 1] + 3 s[6, 4, 2, 2, 1] + s[5, 5, 3, 1, 1]  + s[6, 4, 3, 2]

+ s[5, 5, 2, 2, 1] + s[5, 5, 2, 1, 1, 1] + s[8, 2, 1, 1, 1, 1, 1]

+ s[7, 3, 3, 2]  +    s[7, 3, 1, 1, 1, 1, 1]  +  s[7, 2, 2, 2, 2]

+ s[7, 2, 2, 2, 1, 1] + s[7, 2, 2, 1, 1, 1, 1] + s[6, 3, 3, 2, 1]

+ s[6, 3, 3, 1, 1, 1] + s[6, 4, 1, 1, 1, 1, 1] + s[6, 3, 2, 2, 2]

+ s[6, 3, 2, 2, 1, 1] + s[6, 3, 2, 1, 1, 1, 1] + s[5, 4, 3, 2, 1]

+ s[5, 4, 3, 1, 1, 1]  + s[5, 4, 2, 2, 2]  +  s[5, 4, 2, 2, 1, 1]

+ s[5, 4, 2, 1, 1, 1, 1] + s[9, 2, 2, 1, 1] + s[9, 2, 1, 1, 1, 1]

+ s[9, 2, 2, 2] + s[8, 4, 2, 1] + s[9, 3, 2, 1]
```

Cette formule nous évite par exemple de développer les deux fonctions de Schur sur la base des monômes, d'effectuer le produit puis de repasser sur la base des fonctions de Schur les $80 \times 31 = 2480$ monômes du résultat. Mieux encore, cette formule est indépendante du nombre de variables : par exemple, si le cardinal de l'alphabet avait été égal à 10, le produit des deux fonctions de Schur aurait engendré $11160 \times 20625 = 230175000$ monômes correspondant aux mêmes fonctions de Schur que dans le cas de 4 variables.

Nous avons aussi déjà parlé dans le chapitre premier d'une règle complètement différente donnée par Lascoux et Schützenberger pour calculer le produit de deux fonctions de Schur en utilisant les polynômes de Schubert. La fonction `LRrbyX` réalise cette opération et nous pouvons vérifier qu'elle donne bien le même résultat que précédemment :

```
ACE> LRrSf(s[5,1,1], s[4,2,1,1]) - LRrbyX(s[5,1,1], s[4,2,1,1]);

                                      0
```

2.4 Les opérateurs d'Adams du groupe symétrique

Les *opérations d'Adams* ψ^k de l'anneau $\mathcal{X}(n)$ des caractères du groupe symétrique $\mathfrak{S}_n$ sont définies par :

$$(\psi^k \chi)(w) = \chi(w^k) \quad .$$

Ces opérateurs permettent de calculer les caractères des puissances extérieures $\Lambda^k(\rho)$ ou des puissances symétriques $S^k(\rho)$ d'une représentation ρ de $\mathfrak{S}_n$. En effet, les traces des matrices $\Lambda^k(\rho(w))$, où $w \mapsto \rho(w)$ est la représentation, sont au signe près les coefficients du polynôme caractéristique de $\rho(w)$, qui peuvent donc être calculées à partir des sommes de puissances de ses racines, lesquelles sont précisément les $(\psi^k\chi)(w) = \operatorname{tr}\rho(w^k)$.

Notons $\hat{\psi}^k$ les opérateurs sur les fonctions symétriques correspondant aux opérations d'Adams de $\mathcal{X}(n)$ par l'application caractéristique de Frobenius et soient $\hat{\phi}^k = (\hat{\psi}^k)^*$ leurs adjoints pour le produit scalaire standard. L'intérêt de transporter les calculs dans l'anneau des fonctions symétriques vient de ce que les opérateurs $\hat{\phi}^k$ sont des homomorphismes pour le produit ordinaire des fonctions symétriques. Il est de plus facile de vérifier que $\hat{\phi}^k(p_n) = p_{n/d}^d$, où d est le p.g.c.d. de n et k, d'où le calcul de ces opérateurs donné par la fonction ci-dessous (on note `Iphi(k,f)` pour $\hat{\phi}^k(f)$) :

```
Iphi:=proc(k, sf)
  local
    sfp, i;
  sfp:=Top(sf);
  subs({seq(p.i=(p.(i/igcd(k, i))^igcd(k, i)),
            i=`SYMF/AllValues`(sfp, 'p'))}, sfp)
end;
```

On déduit par exemple de la définition de ces opérateurs et de la formule des caractères de Frobenius que le nombre de racines k-ièmes d'une permutation de type cyclique λ dans $\mathfrak{S}_n$ est égal au produit scalaire :

$$\langle \hat{\phi}^k(h_n)\,,\, p_\lambda \rangle \quad .$$

La fonction `NbRoots` ci-dessous calcule le nombre de racines k-ièmes d'une permutation donnée :

```
Prodp:=proc(lambda)
  convert(map(x->p.x, lambda), `*`)
end;

NbRoots:=proc(k, perm)
  local
    lambda, n;
  lambda:=Perm2CycleType(perm);
  n:=convert(lambda, `+`);
  ScalarSf(Iphi(k, h.n), Prodp(lambda));
end;
```

Par exemple :

```
ACE> NbRoots(3, [2,1,5,6,3,4]);

                                        9
```

On peut maintenant calculer les opérateurs d'Adams eux-mêmes (**Ipsi**(k, f, n)) ainsi que les puissances symétriques (**ISym**) et extérieures (**IExt**) :

```
Ipsi:=proc(k, sf, n)
  local
    sfp, z;
  option remember;
  sfp:=Top(sf);
  convert([seq(ScalarSf(sfp, Iphi(k, Prodp(z)))*Prodp(z)/Zee(z),
             z=ListPart(n))], `+`)
end;

ISym:=proc(k, sf, n)
  local
    i, sfp;
  if (k=1) then
    sf
  elif (k=0) then
    h.n
  else
    sfp:=Top(sf);
    convert([seq(ITensor(Ipsi(i, sfp, n), ISym(k-i, sfp, n)), i=1..k)], `+`)/k
  fi
end;

IExt:=proc(k, sf, n)
  local
    i, sfp;
  if (k=1) then
    sf
  elif (k=0) then
    h.n
  else
    sfp:=Top(sf);
    convert([seq((-1)^(i-1)*ITensor(Ipsi(i, sfp, n), IExt(k-i, sfp, n)),
                 i=1..k)], `+`)/k
  fi
end;
```

On ne connaît que très peu de formules explicites pour ces opérations. Un résultat important donne les puissances extérieures de la “petite équerre” $(n-1, 1)$:

$$\Lambda^k V_{(n-1,1)} = V_{(n-k,1^k)} \quad .$$

Par exemple :

```
ACE> IExt(3, s[5,1], 6):
ACE> Tos(");
                                 s[3, 1, 1, 1]
```

Mais le cas général est beaucoup plus compliqué :

```
ACE> IExt(3, s[3,2,1], 6):
ACE> Tos(");

         4 s[5, 1]  +  7 s[2, 2, 1, 1]  +  7 s[4, 2]  + s[1, 1, 1, 1, 1, 1]

       + 8 s[4, 1, 1] + 4 s[2, 1, 1, 1, 1] + 4 s[2, 2, 2] + 8 s[3, 1, 1, 1]

       + 12 s[3, 2, 1] + s[6] + 4 s[3, 3]
```

2.5 Fonction génératrice des polynômes de Schubert de $\mathfrak{S}_\mathbf{n}$

Les polynômes de Schubert pour le groupe symétrique peuvent être exprimés comme certains coefficients dans l'algèbre des différences divisées [6] ou algèbre nilCoxeter [1]. La fonction génératrice de Fomin et Kirillov est ainsi donnée par :

$$\mathfrak{S}(\mathbf{x}\,;\mathbf{y}) = C_1(x_1)\, C_2(x_2) \ldots C_{n-1}(x_{n-1}) \qquad (2.1)$$

$$= \sum_{\mu \in \mathfrak{S}_n} \mathbb{X}_\mu(\mathbf{x}, \mathbf{y})\ \partial_\mu \quad , \qquad (2.2)$$

dans laquelle $C_i(x) = \Big(1 + (x - y_{n-i})\partial_{n-1}\Big)\Big(1 + (x - y_{n-i-1})\partial_{n-2}\Big) \ldots \Big(1 + (x - y_1)\partial_i\Big)$ et les x_i et y_i commutent avec les ∂_μ.

Voici un programme calculant cette fonction génératrice. La fonction `C` calcule un élément $C_i(x_i)$ en évaluant le produit de tous les termes :

`(1 + (x - y_alphabet[j-i+1])*Transpo(j, n))` ,

dans l'algèbre nilCoxeter (la multiplication dans cette algèbre correspond à la fonction `NcaMult`) :

```
C:=proc(n, i, x, y_alphabet)
  local
    j, r;
  r:=1;
  for j from i to n-1 do
    r := NcaMult((1 + (x - y_alphabet[j-i+1])*Transpo(j, n)), r)
  od;
  r
end;
```

L'appel `Transpo`(j, n) retourne la transposition élémentaire échangeant j et $j+1$ dans $\mathfrak{S}_n$. La fonction génératrice des polynômes de Schubert doubles est alors donnée par les instructions suivantes :

```
An_Schubertfg:=proc(x_alphabet, y_alphabet)
  local
    i, n, r;
  n:=nops(x_alphabet);
  r:=1;
  for i from 1 to n-1 do
    r := NcaMult(r, C(n, i, x_alphabet[i], y_alphabet))
  od;
  r
end;
```

Par exemple, pour $\mathfrak{S}_3$, nous obtenons :

```
ACE> An_Schubertfg([x1, x2, x3], [y1, y2, y3]);

        (x1 - y1) [2, 1, 3] + [1, 2, 3] + (x2 - y1 + x1 - y2) [1, 3, 2]

      + (x1 - y2) (x1 - y1) [3, 1, 2] - (x1 - y1) (- x2 + y1) [2, 3, 1]

      - (x1 - y2) (x1 - y1) (- x2 + y1) [3, 2, 1]
```

2.6 Fonction génératrice des polynômes de Schubert de $\mathrm{B_n}$

Nous avons déjà vu que B_n peut être plongé dans $\mathfrak{S}_{2n}$. Plus précisément, nous rappelons que cela consiste à identifier les générateurs $u_1, \ldots, u_{n-1}$ de l'algèbre nilCoxeter de B_n avec :

$$\begin{array}{rcl} u_{n-1} & = & \partial_1 \partial_{2n-1} \quad , \\ u_{n-2} & = & \partial_2 \partial_{2n-2} \quad , \\ \vdots & \vdots & \vdots \\ u_1 & = & \partial_{n-1} \partial_{n+1} \quad , \end{array}$$

et le générateur supplémentaire u_0 à ∂_n.

La fonction génératrice des polynômes de Schubert de B_n, récemment introduits par Fomin et Kirillov, est donnée par :

$$\mathrm{FK}(\mathbf{x}) = B(x_1)\, B(x_2) \ldots B(x_n)\, C'_1(-x_1)\, C'_2(-x_2) \ldots C'_{n-1}(-x_{n-1}) \tag{2.3}$$

$$= \sum_{\nu \in B_n} \mathfrak{X}_\nu(\mathbf{x})\ u_\nu \tag{2.4}$$

$$= \sum \mathfrak{X}_\nu(\mathbf{x})\ \partial_\mu \quad , \tag{2.5}$$

μ étant l'image de ν dans $\mathfrak{S}_{2n}$, $B(x) = (1 + xu_{n-1}) \ldots (1 + xu_1)\,(1 + xu_0)\,(1 + xu_1) \ldots (1 + xu_{n-1})$ et $C_i'(x) = (1 + xu_{n-1})\,(1 + xu_{n-2}) \ldots (1 + xu_i)$. Tout d'abord, **BnTranspoPalin** réalise ce plongement de B_n dans $\mathfrak{S}_{2n}$ pour une transposition élémentaire. Et puis, l'appel `B`(n, x) calcule $B(x)$ alors que `Bn_C`(n, i, x) calcule quant à lui $C_i'(x)$:

```
B:=proc(n, x)
  local
    j, r;
  r:=1;
  for j from n-1 by -1 to 0 do
    r := BnNcaMult(r, (1 + x*BnPalin2Code(BnTranspoPalin(j, n))))
  od;
  for j from 1 to n-1 do
    r := BnNcaMult(r, (1 + x*BnPalin2Code(BnTranspoPalin(j, n))))
  od;
  r
end;

Bn_C:=proc(n, i, x)
  local
    j, r;
  r:=1;
  for j from i to n-1 do
    r := BnNcaMult((1 + x*BnPalin2Code(BnTranspoPalin(j, n))), r)
  od;
  r
end;

Bn_Schubertfg:=proc(x_alphabet)
  local
    i, n, r;
  n:=nops(x_alphabet);
  r:=1;
  for i from 1 to n do
    r := BnNcaMult(r, B(n, x_alphabet[i]))
  od;
  for i from 1 to n-1 do
    r := BnNcaMult(r, Bn_C(n, i, -x_alphabet[i]))
  od;
  r
end;
```

Par exemple, nous obtenons pour $n = 2$:

```
ACE> map(factor, Bn_Schubertfg([x1, x2]));

                  2
        (x1 + x2)  [0, 2] + (x1 + 2 x2) [0, 1] + (x1 + x2) [1, 0]

              2
         + x2  (x1 + x2) [0, 3] + x2 x1 (x1 + x2) [1, 2] + [0, 0]

              2
         + x2  x1 (x1 + x2) [1, 3] + x2 (x1 + x2) [1, 1]

ACE> BnaCode2Palin(");

            2
  (x1 + x2)  [2, 4, 1, 3] + (x1 + 2 x2) [2, 1, 4, 3] + (x1 + x2) [1, 3, 2, 4]

        2
   + x2  (x1 + x2) [4, 2, 3, 1] + x2 x1 (x1 + x2) [3, 4, 1, 2] + [1, 2, 3, 4]

        2
   + x2  x1 (x1 + x2) [4, 3, 2, 1] + x2 (x1 + x2) [3, 1, 4, 2]
```

Ce petit programme nous a permis de calculer les tables de ces polynômes de Schubert correspondant aux groupes hyperoctaédraux B_3 (voir en annexe) et B_4.

La table suivante reprend le calcul précédent en donnant les $2^2 2!$ coefficients de la fonction génératrice précédemment calculée. La première colonne contient une permutation signée, la deuxième la permutation palindrome correspondante, la troisième le code (au sens de B_n) et la dernière le polynôme de Schubert associé :

$\overline{1}2$	1324	10	$x_1 + x_2$
12	1234	00	1
21	2143	01	$x_1 + 2x_2$
$\overline{2}1$	2413	02	$(x_1 + x_2)^2$
$2\overline{1}$	3142	11	$x_2(x_1 + x_2)$
$1\overline{2}$	4231	03	$x_2^2(x_1 + x_2)$
$\overline{2}\overline{1}$	3412	12	$x_1 x_2(x_1 + x_2)$
$\overline{1}\overline{2}$	4321	13	$x_1 x_2^2(x_1 + x_2)$

Figure 2.1: *Table des polynômes de Schubert correspondant à B_2.*

Il y a d'autres familles de polynômes de Schubert de B_n. Fomin et Kirillov en ont défini une seconde et montré le lien avec une troisième que l'on doit à Billey et Haiman. Pragacz et Ratajski [8] en ont décrit une autre liée à des considérations géométriques et l'on en doit une cinquième à Fulton.

2.7 Développement du q-discriminant

Un des premiers résultats que nous avons obtenu avec la bibliothèque `SYMF` nous a conduit à porter à $n = 8$ l'indice du plus grand q-discriminant effectivement développé sur la base des fonctions de Schur. Le problème est le suivant. Le groupe $GL(n, \mathbb{C})$ agit sur $M_n(\mathbb{C})$ par conjugaison et on a une représentation ρ :

$$\begin{array}{rccc} \rho: & GL(n,\mathbb{C}) & \longrightarrow & \mathrm{Aut}(M_n(\mathbb{C})) \quad , \\ & \rho(g)\, M & \stackrel{\text{def}}{=} & g\, M\, g^{-1} \quad . \end{array}$$

La représentation ρ est appelée représentation adjointe. Le caractère de ρ est entièrement déterminé par ses valeurs sur les matrices diagonales. En tant que fonction symétrique des valeurs propres $x_1, \ldots, x_n$ de la matrice g, il est défini par :

$$\mathrm{tr}\ \rho \begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{pmatrix} = \Big(\sum_i x_i\Big)\Big(\sum_j \frac{1}{x_j}\Big) = \sum_{i,j} \frac{x_i}{x_j} \quad .$$

En effet, chaque matrice E_{ij} avec un 1 en ligne i colonne j est un vecteur propre de valeur propre x_i/x_j. On peut par ailleurs considérer ses puissances symétriques $S^k(\rho)$ ou extérieures $\Lambda^k(\rho)$. Plus précisément, si on s'intéresse à la fonction génératrice des caractères des puissances symétriques de ρ :

$$\sum_{k\geq 0} q^k \text{caractère}\Big(S^k(\rho)\Big) \quad ,$$

on obtient une certaine fonction symétrique :

$$\prod_{i,j} \frac{1}{1 - q\frac{x_i}{x_j}} \quad ,$$

que l'on sait exprimer au moyen des fonctions de Hall-Littlewood. Dans le cas des puissances extérieures de ρ, on ne sait pas dire grand chose de la fonction génératrice si ce n'est qu'elle est égale à :

$$\prod_{i,j} (1 + q\frac{x_i}{x_j}) \quad .$$

Ceci motive donc l'étude du q-discriminant défini par :

$$D_n(q) := \prod_{i,j=1}^{n} (x_i + q x_j) \quad ,$$

dont la décomposition en fonctions de Schur donne la décomposition en irréductibles des puissances extérieures de la représentation adjointe. A cet effet, nous avons programmé un algorithme récent du à Scharf, Thibon et Wybourne [9] que nous résumons ci-dessous. Notons que le développement du discriminant usuel ($q=-1$) sur la base des fonctions de Schur intervient aussi dans l'étude de l'effet Hall quantique.

Nous supposons connue l'expression de $D_{n-1}(q)$ sur la base des fonctions de Schur :

$$D_{n-1}(q) \;=\; \sum_I c_I\, s_I(x_1,\ldots,x_{n-1}) \quad ,$$

où c_I est un coefficient polynomial en q. Nous calculons le facteur manquant F_n entre $D_n(q)$ et $D_{n-1}(q)$:

$$F_n \;=\; x_n(1+q)\prod_{i=1}^{n-1}(x_i+qx_n)(x_n+qx_i) \quad .$$

En écrivant F_n comme :

$$F_n \;=\; x_n(1+q)\prod_{i=1}^{n-1}(x_i+qx_n)\prod_{i=1}^{n-1}(x_n+qx_i) \quad ,$$

on met en évidence deux blocs (les deux produits) symétriques en $x_1,\ldots,x_{n-1}$. On pose alors $F_n^1(q,x_n)$ l'expression du produit $x_n(1+q)\prod_{i=1}^{n-1}(x_i+qx_n)$ sur la base des fonctions symétriques élémentaires en $x_1,\ldots,x_{n-1}$ et $F_n^2(q,x_n)$ l'expression du produit $\prod_{i=1}^{n-1}(x_n+qx_i)$ sur cette même base. Ces deux éléments sont à coefficients en q et x_n. On a donc :

$$F_n \;=\; F_n^1(q,x_n)\,F_n^2(q,x_n) \quad ,$$

et par suite, $D_n(q)=F_n\,D_{n-1}(q)$ s'écrit :

$$D_n(q) \;=\; \Big(\sum_I c_I\, s_I(x_1,\ldots,x_{n-1})\Big)\,F_n^1(q,x_n)\,F_n^2(q,x_n) \quad .$$

Puisque $F_n^1(q,x_n)$ et $F_n^2(q,x_n)$ ne font intervenir que des fonctions élémentaires, la formule de Pieri nous permet de calculer efficacement le produit avec D_{n-1} de sorte que :

$$D_n(q) \;=\; \sum_I d_I(q,x_n)\, s_I(x_1,\ldots,x_{n-1}) \quad . \tag{2.6}$$

La fin du calcul repose sur les deux propriétés suivantes. Soit Ω_n l'opérateur de symétrisation envoyant $x^I := x_1^{i_1}\ldots x_n^{i_n}$ sur la fonction de Schur s_I. Alors :

$$\Omega_n(D_n) = D_n \quad ,$$

car D_n est déjà symétrique en $x_1,\ldots,x_n$. D'autre part [3] :

$$\Omega_n\Big(x_n^k\, s_I(x_1,\ldots,x_{n-1})\Big) = s_{k,I}(x_1,\ldots,x_n) \quad .$$

En appliquant l'opérateur Ω_n aux deux membres de (2.6), on fait rentrer les puissances résiduelles de x_n dans les fonctions de Schur, ce qui conduit au résultat escompté, c'est-à-dire à l'expression de $D_n(q)$ comme somme de fonctions de Schur à coefficients polynomiaux en q. Par exemple pour $n=1$, $D_n(q)$ est réduit à $(1+q)x_1$ et par conséquent $D_1(q) = (1+q)s_1$. Pour $n=2$, on obtient :

$$D_2(q) \;=\; q(q+1)^2 s_{3,1} + (q^2-q+1)(q+1)^2 s_{2,2} \quad ,$$

et l'on vérifie avec `ACE` que le résultat est exact :

```
ACE> CLG_n(2):
ACE> Tox_n(q*(q+1)^2*s[3,1] + (q^2-q+1)*(q+1)^2*s[2,2]);

          3    3        2   2        3       2    3        2   2        3
         q   (x2  x1 + x2  x1  + x2 x1 ) + 2 q  (x2  x1 + x2  x1  + x2 x1 )

              3        2   2        3      4   2   2     3   2   2       2   2
       + q (x2  x1 + x2  x1  + x2 x1 ) + q  x2  x1  + q  x2  x1  + q x2  x1

           2   2
       + x2  x1

ACE> factor(");
                                   2
                 x2 x1 (q + 1)  (q x2 + x1) (x2 + x1 q)
```

Nous avons écrit un programme traduisant cet algorithme en MAPLE. Nous donnons un exemple de l'utilisation de la fonction principale `Qdisc` de ce programme pour $n = 3$:

```
ACE> Qdisc(3, q);

             4    3    2              2                  3
           (q  - q  + q  - q + 1) (q  - q + 1) (1 + q)  s[3, 3, 3]

           3         3                2   2                 3
        + q  (1 + q)  s[5, 3, 1] + q  (q  - q + 1) (1 + q)  s[5, 2, 2]

           2   2                 3
        + q  (q  - q + 1) (1 + q)  s[4, 4, 1]

              2         2                 3
        + q (q  + 1) (q  - q + 1) (1 + q)  s[4, 3, 2]
```

et donnons en annexe les développements du q-discriminant pour n variant de 1 à 6.

Beware of bugs in the above code;
I have only proved it correct, not tried it.

— ***Donald Knuth.***

Références bibliographiques

[1] S. Fomin and A.N. Kirillov. Combinatorial B_n-analogues of Schubert polynomials, (1994). preprint.

[2] W. Fulton. Flags, Schubert polynomials, degeneracy loci and determinantal formulas. *Duke Math. J.*, 65: 381–420, (1991).

[3] A. Lascoux. Inversion des matrices de Hankel. *Linear Algebra and Appl.*, 129: 77–102, (1990).

[4] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

[5] A. Lascoux and M.P. Schützenberger. Keys and standard bases, in invariant theory and tableaux, (1990).

[6] A. Lascoux and M.P. Schützenberger. Algèbre des différences divisées. *Discrete Math.*, 99: 165–179, (1992).

[7] I.G. Macdonald. *Symmetric functions and Hall polynomials.* Oxford: Clarendon Press, (1995). 2nd ed.

[8] P. Pragacz and J. Ratajski. Formulas for Lagrangian and orthogonal degeneraci loci; the $\widetilde{Q}$-Polynomials Approach. to appear in Compositio Math.

[9] T. Scharf, J.-Y. Thibon, and B.G. Wybourne. Powers of the Vandermonde determinant and the quantum Hall effect. *Journal of Physics A*, 27: 4211–4219, (1994).

[10] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13: 179–191, (1961).

[11] M.P. Schützenberger. Propriétés nouvelles des tableaux de Young. In *Séminaire Delange-Pisot-Poitou*, volume 26, (1977).

[12] J.-Y. Thibon and S. Veigneau. Using ACE, an Algebraic Combinatorics Environment for Maple. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 461–468, Minneapolis, (1996). University of Minnesota.

[13] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 1.0. Rapport 95–14, IGM, (1995).

[14] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 2.0. Rapport, IGM, (1996).

[15] S. Veigneau. SP, a package for Schubert polynomials realized with the computer algebra system Maple. *Journal of Symbolic Computation*, (1996). à paraître.

3

Applications à la géométrie énumérative

Ce qui est tordu ne peut être droit,
ce qui manque ne peut se compter.

Vers le milieu du dix-neuvième siècle, avec Chasles et de Jonquières, est apparue la géométrie énumérative, une branche de la géométrie projective ayant pour objet de calculer le nombre de solutions d'un problème géométrique. Par exemple, il est assez facile de se persuader qu'il existe en général exactement 2 droites de l'espace qui s'appuient sur 4 droites données. Mais il est plus difficile de comprendre pourquoi, comme l'a établi Chasles, il existera en général 3264 coniques tangentes à 5 coniques données dans le plan projectif (sur $\mathbb{C}$). Il n'est d'ailleurs pas évident de formuler ces problèmes correctement. Il faut déjà préciser ce qu'on entend par "en général" et surtout, comment compter les solutions multiples.

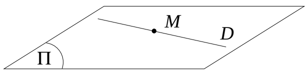
Ces problèmes se ramènent en effet à dénombrer les solutions de systèmes d'équations algébriques et s'il est facile de définir la multiplicité d'une racine d'un polynôme en une variable, le problème multi-dimensionnel est beaucoup plus complexe.

Néanmoins, des raisonnements heuristiques ingénieux, guidés par l'intuition géométrique, ont permis aux géomètres du siècle dernier d'obtenir un grand nombre de résultats corrects. La méthode la plus puissante et la plus intéressante étant sans doute le calcul d'incidence de Schubert, qui à partir d'un petit nombre d'identités obtenues au moyen d'arguments géométriques pas toujours rigoureux, permettait de deviner des résultats spectaculaires par un calcul purement formel. On sait aujourd'hui justifier ce calcul au moyen de théories très sophistiquées. Schubert calculait dans ce qu'on appelle aujourd'hui l'anneau de Chow d'une variété algébrique. Du point de vue algorithmique, ces calculs se ramènent à des manipulations de fonctions symétriques et de polynômes de Schubert.

3.1 Le calcul de Schubert

Dans le cas de problèmes énumératifs concernant seulement des variétés linéaires (droites, plans, ...), on peut le justifier en calculant dans les anneaux d'homologie (ou de cohomologie) d'une grassmannienne ou d'une variété de drapeaux. Ces anneaux se rencontrent dans plusieurs autres contextes, ils sont en particulier importants en théorie des représentations.

Dans ce qui suit, nous nous contentons d'exposer les idées essentielles de la théorie sur le plus petit exemple non trivial, celui de la grassmannienne $G(2,4)$ des 2-plans de $\mathbb{C}^4$, que l'on peut aussi interpréter comme l'ensemble des droites (projectives) de $\mathbb{P}^3 = \mathbb{P}^3(\mathbb{C})$. Fixons un drapeau $V_1 \subset V_2 \subset V_3 \subset V_4 = \mathbb{C}^4$ de sous-espaces vectoriels de $\mathbb{C}^4$, qui définissent respectivement un point M de $\mathbb{P}^3$, une droite D contenant M et un plan Π contenant D.



On peut classer les droites de $\mathbb{P}^3$ de la manière suivante. On indique les dimensions (vectorielles) de leurs intersections avec les sous-espaces du drapeau de référence, c'est-à-dire les dimensions de $W \cap V_i$, où W est le 2-plan de $\mathbb{C}^4$ correspondant :

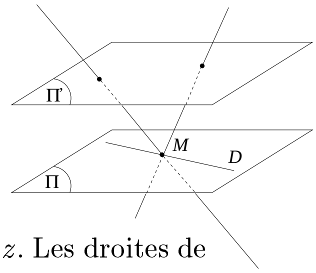
		$M(V_1)$	$D(V_2)$	$\Pi(V_3)$	V_4
non contenues dans Π	celles qui coupent Π mais pas D	0	0	1	2
	celles qui coupent D mais pas M	0	1	1	2
	celles qui coupent D en M	1	1	1	2
contenues dans Π	celles qui coupent D mais pas en M	0	1	2	2
	celles qui coupent D en M mais $\neq D$	1	1	2	2
	la droite D elle-même	1	2	2	2

Si on regarde les suites de dimensions, on voit qu'elles augmentent au plus de 1 à chaque étape et aboutissent toujours à 2. On peut donc indexer les ensembles correspondants par les deux entiers i tels que $\dim(W \cap V_i) = \dim(W \cap V_{i-1}) + 1$:

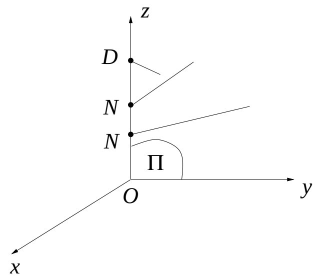
0012	$\longrightarrow$	34	0122	$\longrightarrow$	23
0112	$\longrightarrow$	24	1122	$\longrightarrow$	13
1112	$\longrightarrow$	14	1222	$\longrightarrow$	12

On peut identifier les nouveaux indices à des tableaux colonnes :

- l'ensemble $\begin{array}{|c|}\hline 2\\ 1\\ \hline\end{array}$ est la droite elle-même : c'est un point de $G(2,4)$;
- l'ensemble $\begin{array}{|c|}\hline 3\\ 1\\ \hline\end{array}$ est formé des droites de Π qui passent par M (qui forment un $\mathbb{P}^1$) moins la droite D elle-même. C'est donc $\mathbb{P}^1 - \{D\} = \mathbb{A}^1 \simeq \mathbb{C}$;
- les droites de Π forment un $\mathbb{P}^2$. Si on exclut celles qui passent par M (qui forment un $\mathbb{P}^1$) on obtient que $\begin{array}{|c|}\hline 3\\ 2\\ \hline\end{array}$ est un $\mathbb{P}^2 - \mathbb{P}^1 = \mathbb{A}^2$;



- $\boxed{\substack{4\\1}}$ est l'ensemble des droites passant par M et non contenues dans Π. Si l'on choisit un plan Π' coupant Π selon une droite Δ que l'on peut considérer à l'infini, on voit que les éléments de $\boxed{\substack{4\\1}}$ sont paramétrés par les points de Π' $-\Delta = \mathbb{P}^2 - \mathbb{P}^1 = \mathbb{A}^2$;
- M est le point à l'infini de $D = Oz$. Le plan Π est le plan yOz. Les droites de $\boxed{\substack{4\\2}}$ sont paramétrées par le point $N \in Oz = D - M$ et leur direction qui peut être tout sauf parallèle à yOz, c'est-à-dire $\mathbb{P}^2 - \mathbb{P}^1 = \mathbb{A}^2$. Par suite, $\boxed{\substack{4\\2}} \simeq \mathbb{A}^3$;



- si on prend $\Pi = xOy$, D la droite à l'infini de Π alors $\boxed{\substack{4\\3}}$ est formé des droites de $\mathbb{A}^3$ qui coupent Π transversalement. L'intersection avec Π est paramétrée par $\mathbb{A}^2$ et la direction forme un $\mathbb{P}^2 - \mathbb{P}^1 = \mathbb{A}^2$, donc $\boxed{\substack{4\\3}} \simeq \mathbb{A}^4$.

Les sous-ensembles $\Sigma_{ij} = \boxed{\substack{j\\i}}$ forment une *décomposition en cellules ouvertes* (ils sont isomorphes à des espaces affines et sont 2 à 2 disjoints). On les appelle *cellules de Schubert*. On voit que leurs dimensions sont données par :

$$\dim \boxed{\substack{j\\i}} = (j-2) + (i-1) \quad .$$

Définissons un ordre (dit d'Ehresmann-Bruhat) sur les cellules en convenant que $\Sigma \leq \Sigma'$ si Σ est contenue dans l'adhérence de Σ'. Le *diagramme de Hasse* de cet ordre est donné par la figure 3.1 (page 76). Une autre indexation consiste à poser $\boxed{\substack{j\\i}} = \Omega_\mu$, où μ est le complémentaire à $(2,2)$ de la partition $(j-2, i-1)$. Le poids de μ est alors égal à la codimension de Ω_μ. La figure 3.1 montre aussi l'ordre d'Ehresmann obtenu avec cette nouvelle indexation.

On voit que $\Omega_\mu \subset \overline{\Omega}_\nu \iff \mu \supset \nu$. Les adhérences $\overline{\Omega}_\mu$ des cellules de Schubert sont appelées *variétés de Schubert*. La grassmannienne $G(k,n)$ peut en effet être réalisée comme une sous-variété algébrique de $\mathbb{P}(\Lambda^k\mathbb{C}^n)$ au moyen du *plongement de Plücker* qui à un k-plan Π de vecteurs directeurs $v_1, \ldots, v_k$ associe la droite vectorielle engendrée par $v_1 \wedge \cdots \wedge v_k$ dans $\Lambda^k\mathbb{C}^n$. On peut alors montrer que les images des variétés de Schubert de $G(k,n)$ sont aussi des sous-variétés algébriques de $\mathbb{P}(\Lambda^k\mathbb{C}^n)$, de sorte que les problèmes énumératifs mentionnés ci-dessus reviennent à déterminer le nombre de solutions d'un système d'équations algébriques.

Des résultats généraux de topologie algébrique montrent qu'il est possible (de plusieurs manières) de définir une relation d'équivalence, appelée homologie, sur le $\mathbb{Z}$-module des combinaisons linéaires formelles de sous-variétés d-dimensionnelles (éventuellement singulières) de $G = G(k,n)$. Le $\mathbb{Z}$-module quotient $H_d(G,\mathbb{Z})$ est appelé d-ième groupe d'homologie. Il a pour base les classes d'homologie σ_μ des variétés de Schubert $\overline{\Omega}_\mu$ de dimension d. Il est possible de définir sur $H_*(G,\mathbb{Z})$ une multiplication possédant la propriété suivante :

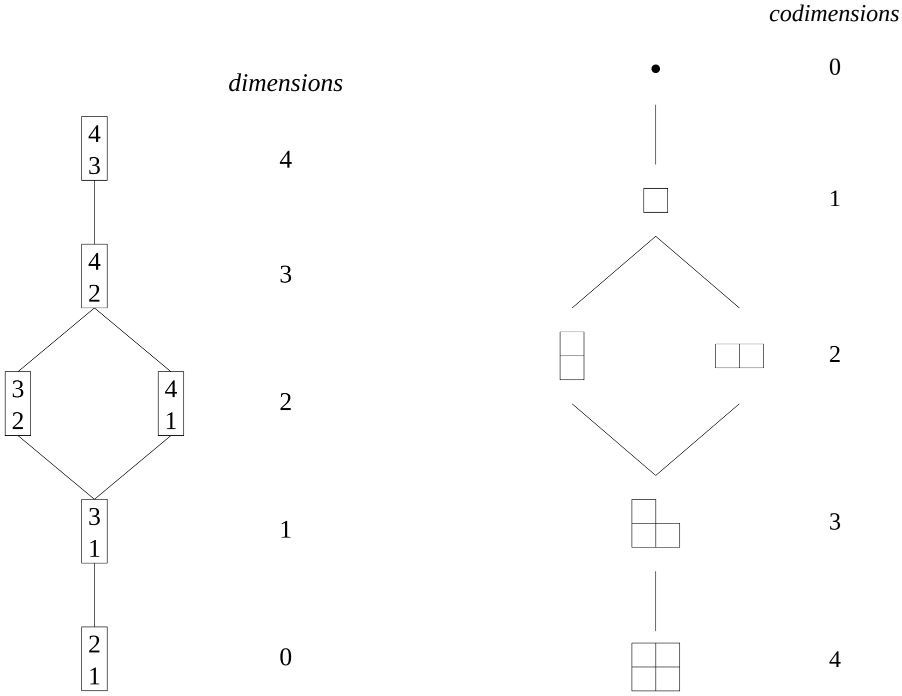


Figure 3.1: *Le diagramme de Hasse de l'ordre de Bruhat pour $G(2,4)$ avec à sa droite son correspondant pour la nouvelle indexation.*

étant données deux sous-variétés U et V de $G(k,n)$ de dimensions complémentaires et se coupant transversalement, on a :

$$[U]\,[V] = m\,\sigma_{(k^{n-k})} \quad ,$$

où $\sigma_{(k^{n-k})}$ est la classe d'un point et m le nombre de points d'intersections de U et V en tenant compte des multiplicités.

Il ne reste plus qu'à préciser que deux variétés de Schubert $\overline{\Omega}_\mu$, $\overline{\Omega}'_\mu$ associées à des drapeaux de référence quelconques sont toujours homologues. Pour revenir à l'exemple du début, on aura ainsi pour $G(2,4)$:

$$\sigma_{(1)}^4 = m\,\sigma_{(2,2)} \quad ,$$

où m est le nombre de droites qui s'appuient sur 4 droites données en position générale. En effet, $\sigma_{(1)}$ est la classe d'homologie de toute sous-variété formée des droites s'appuyant sur une droite fixe.

Avec ces préliminaires, nous pouvons maintenant déterminer la structure multiplicative de $H_*(G(2,4),\mathbb{Z})$. Géométriquement, on a facilement (en représentant σ_μ par le diagramme de μ) :

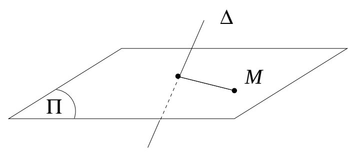
En effet, la première égalité exprime qu'il existe exactement une droite contenue dans deux plans en position générale et la seconde qu'il existe une unique droite passant par deux points distincts. De même :

$$\ydiagram{1,1}\ \ydiagram{2} = \mathbf{0} \quad ,$$

car étant donnés un plan Π et un point M en position générale ($M \notin \Pi$), il n'existe aucune droite de Π qui passe par M. Enfin :

$$\ydiagram{2,1}\ \ydiagram{1} = \mathbf{1}\,\ydiagram{2,2} \quad ,$$

car il n'y a qu'une seule droite d'un plan Π passant par $M \in \Pi$ et coupant une droite générique Δ :



Pour calculer $\ydiagram{1}^{\mathbf{4}}$, on peut procéder en deux temps. Si l'on admet que :

$$\ydiagram{1}^{\mathbf{2}} = \mathbf{a}\,\ydiagram{2} + \mathbf{b}\,\ydiagram{1,1} \quad ,$$

comme le faisaient sans justification les géomètres du siècle dernier, on obtient $\mathbf{a} = \mathbf{b} = 1$ en calculant :

$$\ydiagram{1}^{\mathbf{2}}\ \ydiagram{2} \qquad \text{et} \qquad \ydiagram{1}^{\mathbf{2}}\ \ydiagram{1,1} \quad ,$$

par les formules précédentes. Finalement :

$$\ydiagram{1}^{\mathbf{4}} = \left(\ydiagram{2} + \ydiagram{1,1}\right)^{\mathbf{2}} = \ydiagram{2}^{\mathbf{2}} + \mathbf{2}\,\ydiagram{1,1}\ \ydiagram{2} + \ydiagram{1,1}^{\mathbf{2}} = \mathbf{2}\,\ydiagram{2,2} \quad ,$$

d'où le résultat annoncé.

3.2 Grassmanniennes et fonctions symétriques

Le calcul précédent peut être interprété comme un calcul de polynômes, l'intersection des classes de sous-variétés correspondant à la multiplication de polynômes. Plus précisément, à la sous-variété de Schubert $\overline{\Omega}_\mu$ de $G(k,n)$, on fait correspondre la fonction de Schur s_μ. On peut alors montrer que l'anneau d'homologie de $G(k,n)$ (ou plutôt de cohomologie) s'identifie au quotient :

$$Sym \big/ \mathcal{I}(k,n) \quad ,$$

où $\mathcal{I}(k,n)$ est l'idéal engendré par les fonctions de Schur s_ν telles que ν n'est pas contenue dans la partition (k^{n-k}). La structure multiplicative de $H_*(G(k,n),\mathbb{Z})$ est donc entièrement décrite par la formule de Pieri, qui a d'ailleurs été découverte dans ce cadre et établie par un raisonnement géométrique.

Le programme suivant montre comment réaliser le calcul précédent à l'aide de ACE. La fonction TruncSchur annule les fonctions de Schur de $\mathcal{I}(k,n)$:

```
TruncSchur:=proc(sf, n)
  local
    i;
  if (type(sf, `+`) or type(sf, `*`) or type(sf, `^`)) then
    map(TruncSchur, sf, n)
  elif (type(sf, indexed)) then
    for i from 1 to nops(sf) do if (op(i, sf)>n) then RETURN(0) fi od;
    sf
  else
    sf
  fi
end;

Grassmann:=proc(sf, k, n)
  local
    s_CLGn, res;
  s_CLGn:=_CLGn;
  CLG_n(k);
  res:=TruncSchur(Tos_n(sf), n-k);
  CLG_n(s_CLGn);
  res;
end;
```

On vérifie alors qu'il y a bien 2 droites coupant 4 droites en position générale par la commande suivante :

```
ACE> Grassmann(s[1]^4, 2, 4);

                                  2 s[2, 2]
```

qu'il faut rapprocher du résultat :

$$\square^{4} = \mathbf{2}\,\begin{array}{|c|c|}\hline & \\ \hline & \\ \hline \end{array} \quad .$$

Disposant désormais du formalisme des fonctions symétriques pour résoudre des problèmes géométriques du type de l'exemple précédent, nous pouvons nous demander par exemple combien de droites coupent six plans (en position générale) dans $\mathbb{P}^4$. Pour résoudre ce problème, nous commençons par coder les droites de $\mathbb{P}^4$ (c'est-à-dire les 2-plans de $\mathbb{C}^5$ ou autrement dit les points de la grassmannienne $G(2,5)$) coupant un plan donné. Pour cela, nous calculons les dimensions des intersections de ces droites avec les sous-espaces d'un drapeau de référence $V_1 \subset V_2 \subset V_3 \subset V_4 \subset V_5 = \mathbb{C}^5$:

V_1	V_2	V_3	V_4	V_5
0	0	1	1	2

Par suite, la variété Z des droites de $\mathbb{P}^4$ coupant un plan donné correspond à $\begin{array}{|c|}\hline 5 \\ 3 \\ \hline\end{array}$. Par complémentation de la partition $(5-2, 3-1)$ par rapport à la partition $(3,3)$, on trouve que la classe de Z est :

$$[Z] = \square \quad .$$

Calculer le nombre de droites de $\mathbb{P}^4$ coupant six plans revient donc à chercher le coefficient de $s_{3,3}$ (qui correspond à la classe d'un point) dans le développement de s_1^6 :

$$\square^{\mathbf{6}} = \mathbf{m} \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline\end{array} \quad .$$

Ce nombre qui est le nombre de tableaux standards de forme $(3,3)$ est donné par la formule des équerres (voir formule (7.2), page 148) :

$$\mathbf{m} = \frac{6!}{3 \times 2 \times 1 \times 4 \times 3 \times 2} = 5 \quad .$$

La fonction `Grassmann` permet de vérifier ce résultat :

```
ACE> Grassmann(s[1]^6, 2, 5);

                              5 s[3, 3]
```

3.3 Variétés de drapeaux et polynômes de Schubert

On peut raffiner la construction précédente. La méthode utilisée pour classer les k-plans de $\mathbb{C}^n$ par la suite des dimensions de leurs intersections avec un drapeau de référence s'applique tout aussi bien à des suites de variétés linéaires emboîtées :

$$W_{k_1} \subset \cdots \subset W_{k_r} \quad ,$$

de dimensions $k_1 < \cdots < k_r$, appelées drapeaux complets lorsque $r = n$ et $k_i = i$, et drapeaux incomplets sinon.

Les décompositions en cellules des variétés de drapeaux complets ou incomplets ont été obtenues par Ehresmann, dans sa thèse, publiée en 1934 [2]. En particulier, les cellules de Schubert de la variété $\mathcal{F}_n$ des drapeaux complets de $\mathbb{C}^n$ sont indexées par des suites de tableaux colonnes :

$$\boxed{a_{11}} \quad \boxed{\begin{matrix} a_{22} \\ a_{21}\end{matrix}} \quad \boxed{\begin{matrix} a_{33}\\ a_{32}\\ a_{31}\end{matrix}} \quad \cdots \quad \boxed{\begin{matrix} a_{nn}=n \\ \\ \\ a_{n1}=1\end{matrix}}$$

décrivant comme ci-dessus les dimensions des intersections des éléments du drapeau avec ceux du drapeau de référence. Par construction, ces colonnes sont emboîtées les unes dans les autres, *i.e.* $\{a_{i1},\ldots,a_{ii}\} \subset \{a_{i+1,1},\ldots,a_{i+1,i+1}\}$ de sorte que ces symboles sont en bijection avec les permutations w de $\mathfrak{S}_n$. Il suffit en effet de poser $w_1 = a_{11}$, w_2 égal à l'unique élément de $\{a_{21},a_{22}\} - \{a_{11}\}$ et ainsi de suite.

Les cellules (et les variétés) de Schubert de $\mathcal{F}_n$ sont ainsi indexées par les permutations. De même que les anneaux d'homologie ou de cohomologie de $G(k,n)$, on peut réaliser ceux des variétés de drapeaux comme des quotients d'anneaux de polynômes (non symétriques, cette fois) par un idéal approprié.

Plus précisément, l'anneau de cohomologie de la variété de drapeaux $\mathcal{F}_n$ est isomorphe au quotient $\mathbb{Z}[x_1,\ldots,x_n]/I^+$ où I^+ est l'idéal engendré par les fonctions symétriques sans terme constant en les variables $x_1,\ldots,x_n$ [1].

Pour expliquer la signification géométrique de ce résultat, il faut faire appel à la théorie des fibrés vectoriels, qu'il serait trop long de résumer ici. Disons simplement que tout fibré vectoriel complexe E de rang k sur une base B peut, en tant que variété, se plonger dans $\mathbb{C}^N$, pour un N assez grand et ceci au moyen d'une application dont la restriction à chaque fibre est linéaire. Pour ne pas avoir à se soucier de la valeur précise de N, on le plonge dans $\mathbb{C}^\infty$, muni d'une topologie appropriée. Le plongement $E\xrightarrow{i}\mathbb{C}^\infty$ induit une application $j : B \longrightarrow G(k,\infty)$ qui à un point $p \in B$ associe l'image $i(F_p)$ de la fibre F_p au-dessus de p. On récupère ainsi une application :

$$H^*(G(k,\infty))\xrightarrow{j^*}H^*(B) \quad ,$$

indépendante du plongement. D'après l'étude précédente de la grassmannienne $G(k,n)$, en faisant tendre n vers l'infini, on obtient $H^*(G(k,\infty)) \simeq Sym[x_1,\ldots,x_k]$. Les images $x_i(E) = j^*e_i$ des polynômes symétriques élémentaires sont appelées *classes de Chern* de E. Ce sont des *classes caractéristiques*, c'est-à-dire des invariants du fibré E à valeur dans la cohomologie de la base.

En particulier, un fibré en droites possède une seule classe de Chern $c_1(E)$. Or, la variété de drapeaux est naturellement munie de n fibrés en droites $\mathrm{L}_1,\ldots,\mathrm{L}_n$, dits *tautologiques*. L_1 est le sous-fibré du fibré trivial $\mathcal{F}_n \times \mathbb{C}^n$ dont la fibre au-dessus du point $p = (V_1 \subset V_2 \subset \cdots \subset V_n)$ est simplement la droite V_1. D'une manière générale, on a un sous-fibré E_k de rang k dont la fibre au-dessus de p est V_k. Pour récupérer des fibrés en droites, on pose $L_k = E_k/E_{k-1}$ (*i.e.* la fibre au-dessus de p est la droite V_k/V_{k-1}).

Soit $\gamma_k = c_1(\mathrm{L}_k^\vee) \in H^*(\mathcal{F}_n)$ la première classe de Chern du dual $\mathrm{L}_k^\vee$ de L_k. Alors, comme l'a montré Borel, les γ_k engendrent $H^*(\mathcal{F}_n)$, les fonctions élémentaires des γ_k sont nulles et il n'y a pas d'autres relations non triviales entre ces éléments, de sorte que :

$$H^*(\mathcal{F}_n) = \mathbb{Z}[\gamma_1,\ldots,\gamma_n] \simeq \mathbb{Z}[x_1,\ldots,x_n]\Big/I^+ \quad .$$

Les classes de Chern d'une variété sont par définition celles de son fibré tangent. A titre d'exemple, calculons celles de la variété de drapeaux. Le fibré tangent de la variété de drapeaux admet une suite de composition dont les quotients sont les $L_i^\vee \otimes L_j$, $i < j$. Comme la classe de Chern de $L_i^\vee \otimes L_j$ est $(1 + \gamma_i - \gamma_j)$, la classe de Chern totale du fibré tangent est :

$$c(\mathcal{F}_n) = \prod_{i<j} (1 + \gamma_i - \gamma_j) \quad .$$

La suite d'instructions suivante donne le développement de $c(\mathcal{F}_4)$ dans la base des cycles de Schubert :

```
chern:=proc(n)
  local
    i, j, res;
  res:=1;
  for j to n do
    for i to j-1 do
      res:=res * (1+x.i-x.j)
    od
  od;
  res
end:

ACE> Flag(4):
ACE> chern(4);

                    (1 + x1 - x2) (1 + x1 - x3) (1 + x2 - x3)

                    (1 + x1 - x4) (1 + x2 - x4) (1 + x3 - x4)

ACE> ToX(");

    6 X[2, 3, 1, 4] +  6 X[3, 1, 2, 4]  + 4 X[2, 1, 4, 3]  +  6 X[1, 4, 2, 3]

  + 6 X[3, 2, 1, 4] + 16 X[3, 1, 4, 2]  +  2 X[1, 3, 2, 4] +  2 X[1, 2, 4, 3]

  +   X[1, 2, 3, 4] + 6 X[1, 3, 4, 2]   +  2 X[2, 1, 3, 4] + 22 X[4, 1, 2, 3]

  + 24 X[4, 3, 2, 1] + 22 X[2, 3, 4, 1] + 20 X[2, 4, 1, 3] + 28 X[3, 2, 4, 1]

  + 44 X[3, 4, 1, 2] + 24 X[4, 2, 1, 3] + 24 X[2, 4, 3, 1] + 28 X[4, 1, 3, 2]

  +  6 X[1, 4, 3, 2] + 48 X[4, 2, 3, 1] + 36 X[4, 3, 1, 2] + 36 X[3, 4, 2, 1]
```

De plus, les polynômes de Schubert sont compatibles avec les plongements $H^*(\mathcal{F}_{i_1\cdots i_r}) \hookrightarrow H^*(\mathcal{F}_n)$ des anneaux de cohomologie des variétés de drapeaux incomplets donnés par les fibrations :

$$(V_0 \subset V_1 \subset \cdots \subset V_n) \longrightarrow (V_{i_1} \subset V_{i_1+i_2} \subset \cdots \subset V_{i_1+\cdots+i_r}) \quad .$$

La classe de Chern totale d'une variété de drapeaux incomplets est un facteur de la classe totale. Par exemple, la classe de Chern de la grassmannienne $G(k,n)$ est égale à :

$$\prod_{1\leq i\leq k}\ \prod_{k+1\leq j\leq n}(1+\gamma_i-\gamma_j) \quad .$$

L'exemple suivant donne cette classe pour $G(2,4)$:

```
ACE> Flag(4):
ACE> res:=1:
ACE> for i from 1 to 2 do for j from 3 to 4 do res:=res * (1+x.i-x.j) od od:
ACE> ToX(res);

                7 X[2, 3, 1, 4] + 7 X[1, 4, 2, 3]  + 4 X[1, 3, 2, 4]

                + X[1, 2, 3, 4] + 12 X[2, 4, 1, 3] + 6 X[3, 4, 1, 2]
```

Pour obtenir le résultat dans la base usuelle des fonctions de Schur, on peut utiliser la fonction `X2Y` :

```
ACE> X2Y(");

            7 Y[1, 1, 0, 0] + 7 Y[0, 2, 0, 0]  + 4 Y[0, 1, 0, 0]

            + Y[0, 0, 0, 0] + 12 Y[1, 2, 0, 0] + 6 Y[2, 2, 0, 0]
```

Lorsque le code de la permutation a la forme $I = J.0^{n-k}$, où J est une partition, le polynôme de Schubert $\mathbf{Y}_I$ correspond au cycle de Schubert s_J de $G(k,n)$. Les propriétés des classes de Chern des variétés de drapeaux sont détaillées dans [6] et [5].

3.4 Degrés projectifs des variétés de Schubert

Le degré projectif d'une sous-variété $X \subset \mathbb{P}^M$ de codimension k est le nombre de points d'intersection d'un k-plan générique avec X. Dans le cas d'une sous-variété de Schubert $\mathbf{X}_\mu$ de la variété de drapeaux $X = \mathcal{F}(\mathbb{C}^n)$ plongée dans $\mathbb{P}^M$ (avec $M = 2^N - 1$ et $N = \frac{n(n-1)}{2}$, dimension de X) par le plongement de Plücker, il est égal au coefficient de σ_ω (classe d'un point) dans le produit $\sigma_\mu h^{N-l(\mu)}$ où h est la classe d'une section hyperplane de X. L'image du fibré vectoriel inversible tautologique de $\mathbb{P}^M$ est $L_1^{\otimes n-1} \otimes \cdots \otimes L_{n-2}^{\otimes 2} \otimes L_{n-1}$, dont la première classe de Chern est $h = (n-1)x_1 + (n-2)x_2 + \cdots + x_{n-1}$. C'est donc la classe h d'une section hyperplane. Les fonctions suivantes permettent de calculer ces degrés :

```
h:=proc(n)
  local
    i;
  option remember;
  convert([seq((n-i)*x.i, i=1..n-1)], `+`)
end;

ProjDegree:=proc(perm)
  local
    res, n, sFlag, d;
  n:=nops(perm);
  d:=n*(n-1)/2 - Perm2Length(perm);
  sFlag:=_Sn;
  Flag(n);
  res:=subs(X[op(LastPerm(n))]=1, ToX(h(n)^d*X[op(perm)]));
  if (sFlag<>-1) then
    Flag(sFlag)
  else
    Flag()
  fi;
  res
end;
```

Par exemple, le degré projectif de la variété de Schubert $\overline{\Omega}_{2,1,4,3}$ de $\mathcal{F}_4$ est 78 comme le montre le résultat suivant :

```
ACE> ProjDegree([2,1,4,3]);

                                     78
```

cela revenant à effectuer le calcul :

```
ACE> ToX((3*x1 + 2*x2 + x3)^4 * X[2,1,4,3]);

                              78 X[4, 3, 2, 1]
```

Finalement, nous donnons les degrés projectifs des 24 variétés de Schubert de $\mathcal{F}_4$, classées par codimension :

```
ACE> ListPerm(4, 'level');

    [[[1, 2, 3, 4]], [[1, 2, 4, 3], [1, 3, 2, 4], [2, 1, 3, 4]],

     [[1, 4, 2, 3], [1, 3, 4, 2], [3, 1, 2, 4], [2, 1, 4, 3], [2, 3, 1, 4]],

     [[4, 1, 2, 3], [1, 4, 3, 2], [3, 1, 4, 2],

                                  [2, 4, 1, 3], [2, 3, 4, 1], [3, 2, 1, 4]],

     [[4, 1, 3, 2], [3, 4, 1, 2], [4, 2, 1, 3], [2, 4, 3, 1], [3, 2, 4, 1]],

     [[4, 3, 1, 2], [4, 2, 3, 1], [3, 4, 2, 1]], [[4, 3, 2, 1]]]

ACE> map(proc(l) map(ProjDegree, l) end, ");

      [[720], [220, 280, 220], [46, 48, 48, 78, 46], [6, 16, 14, 12, 6, 16],

       [3, 2, 3, 3, 3], [1, 1, 1], [1]]
```

3.5 Classe de Todd de la variété de drapeaux

Une autre classe caractéristique importante est la classe de Todd, qui intervient dans les généralisations du théorème de Riemann-Roch donnés par Hirzebruch et Grothendieck, ainsi que dans le théorème de l'index d'Atiyah-Singer. Ici encore on appellera classe de Todd d'une variété celle de son fibré tangent.

La classe de Todd d'un fibré inversible de classe de Chern $1 + x$ est par définition égale à $x/(1 - e^{-x})$. La classe de Todd du fibré tangent de $\mathcal{F}(\mathbb{C}^n)$ est donc égale à :

$$\prod (x_i - x_j)/(1 - e^{x_j - x_i}) \quad .$$

Comme cette classe est un élément de l'anneau de cohomologie, elle est tronquée à l'ordre N et donc égale au polynôme :

$$1 + \frac{h}{1} + \cdots + \frac{h^N}{N!} \quad ,$$

où $N = \binom{n}{2}$ est la dimension de $\mathcal{F}(\mathbb{C}^n)$. La fonction suivante permet de calculer une classe de Todd :

```
Todd:=proc(n)
  local
    pol, hk, sFlag, k;
  pol:=1;
  hk:=1;
  sFlag:=_Sn;
```

```
  Flag(n);
  for k from 1 to n*(n-1)/2 do
    hk:=ToX(hk*h(n));
    pol:=pol + hk/factorial(k)
  od;
  if (sFlag<>-1) then
    Flag(sFlag)
  else
    Flag()
  fi;
  pol
end;
```

Nous illustrons ce calcul pour $\mathcal{F}(\mathbb{C}^4)$:

```
ACE> Todd(4);

  1 +  X[2, 1, 3, 4]  +  X[1, 3, 2, 4]  +  X[1, 2, 4, 3]  +   3/2 X[3, 1, 2, 4]

    + 3/2 X[2, 3, 1, 4] + X[2, 1, 4, 3] + 3/2 X[1, 4, 2, 3] + 3/2 X[1, 3, 4, 2]

    + 8/3 X[4, 1, 2, 3] + X[3, 2, 1, 4] +  2 X[3, 1, 4, 2]  + 7/3 X[2, 4, 1, 3]

                                                              23
    + 8/3 X[2, 3, 4, 1] + X[1, 4, 3, 2] +  2 X[4, 2, 1, 3] + ---- X[4, 1, 3, 2]
                                                              12

                              23
    + 13/4 X[3, 4, 1, 2] + ---- X[3, 2, 4, 1] + 2 X[2, 4, 3, 1] + X[4, 3, 2, 1]
                              12

    + 11/6 X[4, 3, 1, 2] + 7/3 X[4, 2, 3, 1] + 11/6 X[3, 4, 2, 1]
```

3.6 Polynômes de Grothendieck

Une manière plus précise de décrire la géométrie d'une variété X consiste à remplacer la cohomologie par l'anneau $K(X)$ des classes d'isomorphismes de fibrés vectoriels de base X, ou mieux, lorsque X est un espace homogène d'un groupe G, par la K-théorie équivariante $K_G(X)$, anneau des classes de G-isomorphismes de fibrés de base X.

Dans le cas où $X = \mathcal{F}(\mathbb{C}^n)$ est la variété de drapeaux et $G = GL(n, \mathbb{C})$, $K_G(X) = \mathbb{Z}[x_1^{\pm 1}, \ldots, x_n^{\pm 1}]$ et les variétés de Schubert sont représentées par des polynômes G_μ appelés polynômes de Grothendieck [4]. Ces polynômes sont les classes des faisceaux structuraux des variétés de Schubert.

Dans l'anneau de cohomologie, la classe de la variété de Schubert de dimension 0 (c'est-à-dire qui correspond à un point) est égale à $\gamma_1^{n-1} \ldots \gamma_{n-1}$. Similairement, dans l'anneau de Grothendieck, la classe d'un point est égale à $\prod(1-1/x_i)^{n-i}$, où cette fois-ci x_i est la classe du fibré inversible tautologique L_i mentionné plus haut.

On peut en fait associer plus généralement à un point le polynôme :

$$G_\omega := \prod_{i+j\leq n} (1-\frac{y_j}{x_i}) \quad ,$$

en introduisant un deuxième ensemble de variables. Ce polynôme est donné par la fonction `GrothMax` :

```
GrothMax:=proc(n)
  local
    gm, i, j;
  option remember;
  gm:=1;
  for i from 1 to n-1 do
    for j from 1 to n-i do
      gm:=gm*(1 - (y.j)/(x.i))
    od
  od;
  gm
end;
```

Sa spécialisation $y_1=1,\dots,y_n=1$ est la classe d'un point dans l'anneau de Grothendieck de $\mathcal{F}(\mathbb{C}^n)$. Les polynômes représentant les variétés de Schubert s'obtiennent comme image de G_ω par les produits de différences divisées isobares $1+\pi_i$ (produits dans l'algèbre idCoxeter). Plus précisément, si $s_{i_1}\dots s_{i_k}$ est une décomposition réduite de $\mu^{-1}\omega$, alors :

$$G_\mu := \mathtt{IdcaOnPol}((1+\pi_{i_1})\cdots(1+\pi_{i_k}), G_\omega) \quad .$$

La fonction suivante permet de calculer ces polynômes :

```
Groth:=proc(perm)
  local
    n,
    wperm;
  n:=nops(perm);
  wperm:=MultPerm(InvPerm(perm), LastPerm(n));
  simplify(IdcaOnPol(Demazure(wperm), GrothMax(n)))
end;
```

La fonction `TableGG` retourne la table de tous les polynômes de Grothendieck correspondant à un groupe symétrique donné $\mathfrak{S}_n$:

```
TableGG:=proc(n)
  local
    p, t;
  t:=table();
  for p in ListPerm(n) do
    t[p]:=Groth(p)
  od;
  eval(t)
end;
```

L'instruction suivante illustre ce calcul pour $\mathfrak{S}_3$:

```
ACE> TableGG(3);

                table([
                                    (x1 - y2) (- x1 + y1)
                    [3, 1, 2] = - ---------------------
                                               2
                                             x1
                                  - x1 + y1
                    [2, 1, 3] = - ---------
                                     x1
                                (- x1 + y1) (- x2 + y1)
                    [2, 3, 1] = -----------------------
                                         x1 x2
                                (- x1 + y1) (x1 - y2) (- x2 + y1)
                    [3, 2, 1] = ---------------------------------
                                                2
                                              x1  x2
                    [1, 2, 3] = 1
                                  - x2 x1 + y2 y1
                    [1, 3, 2] = - ---------------
                                       x1 x2
                ])
```

En spécialisant $y_i \longrightarrow 1$ et $x_i \longrightarrow 1/(1-x_i)$, les polynômes de Grothendieck deviennent égaux à $\mathbf{X}_\mu$ plus des termes de plus hauts degrés. La fonction suivante permet de mettre en évidence cette propriété :

```
TableGG2X:=proc(n)
  local
    i, t;
  t:=TableGG(n);
  t:=subs(seq(y.i=1,i=1..n-1), eval(t));
  t:=subs(seq((x.i)=1/(1-(x.i)),i=1..n-1), eval(t));
  map(x2X, map(simplify, eval(t)))
end;
```

que nous illustrons pour $\mathfrak{S}_3$ (le terme supplémentaire $-\mathbf{X}_{2,3,1}$ est de degré supérieur à celui de $\mathbf{X}_{1,3,2}$) :

```
ACE> TableGG2X(3);

                   table([
                       [3, 1, 2] = X[3, 1, 2]
                       [2, 1, 3] = X[2, 1, 3]
                       [2, 3, 1] = X[2, 3, 1]
                       [3, 2, 1] = X[3, 2, 1]
                       [1, 2, 3] = X[1, 2, 3]
                       [1, 3, 2] = X[1, 3, 2] - X[2, 3, 1]
                   ])
```

3.7 Polynômes clés

Les polynômes clés sont les images des monômes dominants $x_1^{i_1}x_2^{i_2}\ldots, i_1 \geq i_2 \geq \cdots$, par les opérateurs de Demazure que nous venons de voir. Le programme suivant permet de calculer ces polynômes. La fonction `ListDom` retourne la liste de toutes les partitions plus petites composante à composante que la partition $(n-1, n-2, \ldots, 0)$:

```
ListDom:=proc(n)
  local
    i;
  map(proc(l, n)
        [op(l), 0$(n-nops(l))]
      end, ListPartIn([seq(-i, i=-(n-1)..-1)]), n)
end;
```

L'appel `KeyPol`(*cod*) calcule le polynôme clé correspondant à la permutation dont le code est passé en argument :

```
KeyPol:=proc(cod)
  local
    i, j, pos, tmp, perm, rcod;
  rcod:=sort(cod,proc(x,y) evalb(x>=y) end);
  perm:=NULL;
  tmp:=rcod;
  for j from 1 to nops(cod) do
    member(cod[j], tmp, 'pos');
    perm:=perm, pos;
    tmp:=subsop(pos=-1, tmp);
```

```
  od;
  perm:=[perm];
  IdcaOnPol(Demazure(InvPerm(perm)),
            convert([seq(x.i^(rcod[i]), i=1..nops(cod))],`*`))
end;
```

La fonction suivante retourne la table de tous les polynômes clés d'un groupe symétrique donné, la table étant indicée par les codes des permutations :

```
TableK:=proc(n)
  local
    cod, t;
  t:=table();
  for cod in Interval(LastPerm(n)) do
    t[cod]:=KeyPol(cod)
  od;
  eval(t)
end;
```

Voici un exemple pour $\mathfrak{S}_3$:

```
ACE> TableK(3);

                      table([
                                        2
                          [2, 0, 0] = x1
                          [1, 1, 0] = x2 x1
                          [1, 0, 0] = x1
                          [0, 1, 0] = x2 + x1
                          [0, 0, 0] = 1
                                        2
                          [2, 1, 0] = x1  x2
                      ])
```

Les polynômes de Schubert et les polynômes clés coïncident dans le cas des permutations vexillaires. La fonction suivante retourne la table des polynômes clés dans la base Schubert, indicée par des codes :

```
TableKY:=proc(n)
  eval(map(ToY, TableK(n)))
end;
```

Puisque la seule permutation non vexillaire de $\mathfrak{S}_4$ est $w := [2,1,4,3]$ (dont le code est $[1,0,1,0]$), nous n'avons qu'un terme irrégulier dans la table suivante :

```
ACE> TableKY(4):

                table([
                  [3, 0, 1, 0] = Y[3, 0, 1, 0]
                  [3, 0, 0, 0] = Y[3, 0, 0, 0]
                  [2, 2, 1, 0] = Y[2, 2, 1, 0]
                  [2, 2, 0, 0] = Y[2, 2, 0, 0]
                  [2, 1, 1, 0] = Y[2, 1, 1, 0]
                  [2, 1, 0, 0] = Y[2, 1, 0, 0]
                  [2, 0, 1, 0] = Y[2, 0, 1, 0]
                  [2, 0, 0, 0] = Y[2, 0, 0, 0]
                  [0, 2, 1, 0] = Y[0, 2, 1, 0]
                  [1, 2, 1, 0] = Y[1, 2, 1, 0]
                  [0, 2, 0, 0] = Y[0, 2, 0, 0]
                  [1, 2, 0, 0] = Y[1, 2, 0, 0]
                  [3, 2, 1, 0] = Y[3, 2, 1, 0]
                  [0, 1, 1, 0] = Y[0, 1, 1, 0]
                  [1, 1, 1, 0] = Y[1, 1, 1, 0]
                  [0, 1, 0, 0] = Y[0, 1, 0, 0]
                  [1, 1, 0, 0] = Y[1, 1, 0, 0]
                  [0, 0, 1, 0] = Y[0, 0, 1, 0]
                  [3, 2, 0, 0] = Y[3, 2, 0, 0]
                  [1, 0, 1, 0] = - Y[2, 0, 0, 0] + Y[1, 0, 1, 0]
                  [0, 0, 0, 0] = Y[0, 0, 0, 0]
                  [3, 1, 1, 0] = Y[3, 1, 1, 0]
                  [1, 0, 0, 0] = Y[1, 0, 0, 0]
                  [3, 1, 0, 0] = Y[3, 1, 0, 0]
                ])
```

3.8 Anneau de cohomologie d'une grassmannienne relative

Comme première application des méthodes décrites à la fin du chapitre précédent pour le calcul dans l'anneau des polynômes, nous mentionnons l'anneau de cohomologie d'une variété grassmannienne relative. Plus précisément, soient A et B deux alphabets disjoints $A := \{x_1, \ldots, x_m\}$ et $B := \{x_{m+1}, \ldots, x_n\}$ de cardinaux respectifs m et $n-m$. Nous désignerons par $A+B$ l'alphabet $\{x_1, \ldots, x_m, x_{m+1}, \ldots, x_n\}$.

L'anneau mentionné ci-dessus est l'espace des polynômes en $x_1, \ldots, x_n$ qui sont symétriques en $x_1, \ldots, x_m$ d'un côté et en $x_{m+1}, \ldots, x_n$ de l'autre. Cet espace est un module libre sur l'anneau des polynômes symétriques $Sym[x_1, \ldots, x_n]$. Il a une base canonique (les cycles de Schubert) de cardinal $\binom{n}{m}$, consistant en les fonctions de Schur indicées par les partitions plus petites composante à composante que $(n-m)^m$.

Les calculs que nous avons fait précédemment étaient dans l'anneau des polynômes modulo l'idéal I^+ engendré par les polynômes symétriques sans terme constant. Le cas des grassmanniennes relatives est donc plus précis et redonne le cas précédent en annulant les fonctions de Schur en $x_1, \ldots, x_m$ d'indice non contenu dans $(n-m)^m$. En effet, ces fonctions appartiennent à l'idéal engendré par les fonctions symétriques en $x_1, \ldots, x_n$.

Par exemple, dans le cas $m = 1$, la base consiste en $x_1^0, x_1^1, \ldots, x_1^{n-1}$ et la décomposition d'un polynôme dans cette base est donnée par l'algorithme d'Euclide. Pour un m plus général, le produit de deux fonctions de Schur appartenant à la base, donné par la règle de Littlewood-Richardson [9], impliquera des partitions qui ne seront pas plus petites que $(n-m)^m$.

Cependant, on peut exprimer ce produit de fonctions de Schur en termes de polynômes de Schubert $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$. Ces polynômes doivent être symétriques en $x_1, \ldots, x_m$. Parce que les permutations qui indicent ces polynômes sont dans $\mathfrak{S}_n$, les fonctions de Schur correspondantes sont indicées par des partitions plus petites ou égales à $(n-m)^m$, c'est-à-dire appartiennent à la base canonique.

Dit autrement, les méthodes développées pour calculer dans l'anneau des polynômes à plusieurs variables permettent de décomposer n'importe quel élément de l'anneau de cohomologie dans la base des cycles de Schubert. Par exemple, pour $m = 3$, $n = 5$, le carré de la fonction de Schur $s_{21}(x_1, x_2, x_3)$ est égal à : $e_4 s_2 - e_3 s_{21} + e_1^2 s_{22} + (e_4 - e_3 e_1) s_{11} + (e_4 e_1 + e_5) s_1 + (e_1^2 - e_2) s_{211} - (e_2 e_1 + e_3) s_{111} + s_{222} + 2 e_1 s_{221} + e_5 e_1$ comme le montrent les instructions suivantes :

```
ACE> FM_n(5):
ACE> CLG_n(3):
ACE> sf := s2x_n(s[2,1]);

              2        2                          2       2     2           2
      sf := x3  x2 + x3  x1 + 2 x3 x2 x1 + x3 x2  + x3 x1  + x2  x1 + x2 x1

ACE> x2XfixScal(sf^2, collect);

        s[2, 1, 1, 1, 1] X[1, 2, 3, 4, 5]

      - (- 2 s[1, 1, 1, 1, 1] - s[2, 1, 1, 1]) X[1, 2, 4, 3, 5]

      + s[1, 1, 1, 1] X[1, 2, 5, 3, 4] - s[2, 1, 1] X[1, 3, 4, 2, 5]

      - s[1, 1, 1] X[1, 3, 5, 2, 4] - (2 s[1, 1, 1] + s[2, 1]) X[2, 3, 4, 1, 5]

      + (s[2] + s[1, 1]) X[1, 4, 5, 2, 3] + s[2] X[2, 3, 5, 1, 4]

      + 2 s[1] X[2, 4, 5, 1, 3] + X[3, 4, 5, 1, 2]
```

alors que la règle de Littlewood-Richardson donne $s_{42}+s_{411}+s_{33}+2s_{321}+s_{222}$ qui implique des partitions sortant de $(2,2,2)$:

```
ACE> CLG_n(3):
```

```
ACE> Tos_n(s[2,1]^2);

        s[4, 2] + s[4, 1, 1] + s[3, 3] + 2 s[3, 2, 1] + s[2, 2, 2]
```

On remarquera que modulo l'idéal I^+, ces deux expressions se réduisent à s_{222} qui est la classe d'un point de $G(2,5)$.

Références bibliographiques

[1] A. Borel. Sur la cohomologie des espaces fibrés principaux et des espaces homogènes des groupes de Lie compacts. *Annals of Mathematics*, 57: 115–207, (1953).

[2] C. Ehresmann. Sur la topologie de certains espaces homogènes. *Annals of Mathematics*, 35: 396–443, (1934).

[3] A. Lascoux. Classes de Chern des variétés de drapeaux. *C.R. Acad. Sci. Paris*, 295: 393–398, (1982).

[4] A. Lascoux. Anneau de Grothendieck de la variété de drapeaux. *in The Grothendieck Festschrift*, 3: 1–34, (1990).

[5] A. Lascoux, B. Leclerc, and J.-Y. Thibon. Twisted action of the symmetric group on the cohomology of a flag manifold. *Banach Center Publications*, 36: 111–124, (1996).

[6] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

[7] A. Lascoux and M.P. Schützenberger. Structure de Hopf de l'anneau de cohomologie et de l'anneau de Grothendieck d'une variété de drapeaux. *C.R. Acad. Sci. Paris*, 295: 629–633, (1982).

[8] A. Lascoux and M.P. Schützenberger. Symmetry and flag manifolds. *Springer L.N.*, 996: 118–144, (1983).

[9] I.G. Macdonald. *Notes on Schubert polynomials*, volume 6. Publ. LACIM, Montréal, (1991).

[10] J.-Y. Thibon. Fonctions symétriques et opérations d'Adams de l'anneau de Grothendieck d'un groupe symétrique. *Comptes Rendus de l'Académie des Sciences*, 313: 829–832, (1991).

[11] J.-Y. Thibon and S. Veigneau. Using ACE, an Algebraic Combinatorics Environment for Maple. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 461–468, Minneapolis, (1996). University of Minnesota.

[12] S. Veigneau. ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 2.0. Rapport, IGM, (1996).

Conclusion

Il est moins facile de mesurer la valeur logique d'une conclusion
en matière de sentiment, d'amour, de foi,
qu'il n'est facile de mesurer la valeur logique d'une conclusion
en matière de géométrie.

— ***Antonio Fogazzaro***, *Le Saint.*

La version 2.0 de ACE devra subir de profondes modifications pour que la prochaine version de l'environnement voie le jour. Il faudra simplifier en limitant le nombre de fonctions de chaque bibliothèque afin de ne pas fondre les algorithmes primordiaux au milieu de fonctions plus techniques que mathématiques. Il faudra revoir l'organisation de l'ensemble car les changements apportés à la version 2.0 vont dans le bon sens mais ne seront pas suffisants face à l'explosion des développements envisagés. Autrement dit, l'outil commence à souffrir de sa taille alors que cela devrait être un atout.

Au-delà des problèmes mathématiques et algorithmiques qui sont l'objet de toute notre attention pour obtenir un niveau de performance respectable, nous nous heurtons à des problèmes de gestion du projet. La solution passe notamment par une automatisation des tâches de création des différentes distributions en fonction de la plate-forme et de la version de MAPLE V et une simplification de la phase d'installation de l'ensemble.

La prochaine version de ACE devra paraître plus simple, tout en étant plus évoluée, complexe et complète. Elle intègrera d'autres familles de groupes et algèbres classiques ou quantiques, l'algèbre de Lie libre, l'algèbre de Virasoro, les fonctions de Hall-Littlewood et de Macdonald, ainsi qu'une bibliothèque destinée à la géométrie algébrique.

In seeking the unattainable,
simplicity only gets in the way.

— ***Epigrams in Programming***, *ACM SIGPLAN Sept. 1982.*

Deuxième Partie

Polynômes de Schubert

4

Introduction aux polynômes de Schubert

L'homme qui apprend doit croire ;
celui qui sait doit examiner.

— ***Roger Bacon.***

Dans la première partie de ce mémoire, nous avons eu l'occasion de présenter les polynômes de Schubert et d'indiquer quelques unes de leurs interprétations. Dans ce chapitre, nous rassemblons les définitions algébriques et combinatoires nécessaires à la compréhension du chapitre suivant, qui présente nos contributions à la théorie.

Soit $\mathbf{x} := \{x_1, \ldots, x_n\}$ un alphabet totalement ordonné de variables. On note $s_i, i = 1, \ldots, n-1$ les transpositions élémentaires vues comme opérateurs sur l'anneau $\mathbb{Z}[\mathbf{x}]$ qui échangent x_i et x_{i+1} sans toucher aux autres variables. Nous avons déjà vu précédemment que les transpositions élémentaires satisfont les relations de tresse suivantes :

$$s_i s_j = s_j s_i \qquad \text{si } |i-j| > 1 \quad , \tag{4.1}$$

$$s_i s_j s_i = s_j s_i s_j \qquad \text{si } j = i+1 \quad . \tag{4.2}$$

Nous avons aussi besoin des opérateurs $\partial_i, i = 1, \ldots, n-1$ agissant sur l'anneau des polynômes de la manière suivante :

$$\mathbb{Z}[x_1, \ldots, x_n] \ni \quad p \longrightarrow \partial_i(p) := \frac{p - s_i\, p}{x_i - x_{i+1}} \quad , \tag{4.3}$$

où $s_i\, p$ dénote le polynôme p dans lequel les variables x_i et x_{i+1} ont été échangées. Ces opérateurs linéaires, introduits par Newton et appelés *différences divisées de Newton* ont été remis à l'ordre du jour en 1973 par Bernstein, Gelfand & Gelfand [2] et Demazure [1] qui ont observé qu'ils vérifiaient, tout comme les transpositions élémentaires, les relations de tresse :

$$\partial_i \partial_j = \partial_j \partial_i \qquad \text{si } |i-j| > 1 \quad , \tag{4.4}$$

$$\partial_i \partial_j \partial_i = \partial_j \partial_i \partial_j \qquad \text{si } j = i+1 \quad . \tag{4.5}$$

Il est bien connu par ailleurs que le groupe symétrique $\mathfrak{S}_n$ de degré n est engendré par les transpositions élémentaires $s_i, i = 1, \ldots, n-1$, de sorte que pour toute permutation $\mu \in \mathfrak{S}_n$, on peut trouver une expression de μ comme un produit $s_{i_1} \ldots s_{i_k}$ de transpositions élémentaires. Cette décomposition n'est en général pas unique à cause des relations (4.1), (4.2) et des relations supplémentaires $s_i^2 = 1, i = 1, \ldots, n-1$. La décomposition est dite *réduite* lorsque k est exactement le nombre d'inversions de la permutation μ, qui est dit *longueur* de μ.

Nous rappelons aussi qu'il y a deux ordres naturels sur le groupe symétrique qui en font un ensemble partiellement ordonné avec une fonction de rang qui est la longueur des permutations (voir chapitre 1). Le permutoèdre est le graphe ayant les permutations comme sommets et des arêtes entre toutes les paires $(w, ws), w \in \mathfrak{S}_n$ et s transposition simple. C'est le diagramme de Hasse d'un ordre sur le groupe symétrique, appelé ordre faible. On note t_{ij} la transposition (i, j). Le Bruhatoèdre est le graphe ayant les permutations comme sommets et des arêtes entre toutes les paires $(w, wt), w \in \mathfrak{S}_n$, t transposition telle que $|l(wt) - l(w)| = 1$. L'ordre correspondant est l'ordre de Bruhat, appelé aussi ordre fort. La figure 4.1 montre le permutoèdre de $\mathfrak{S}_3$ où l'on a inscrit des codes au lieu des permutations et sur sa droite le Bruhatoèdre correspondant :

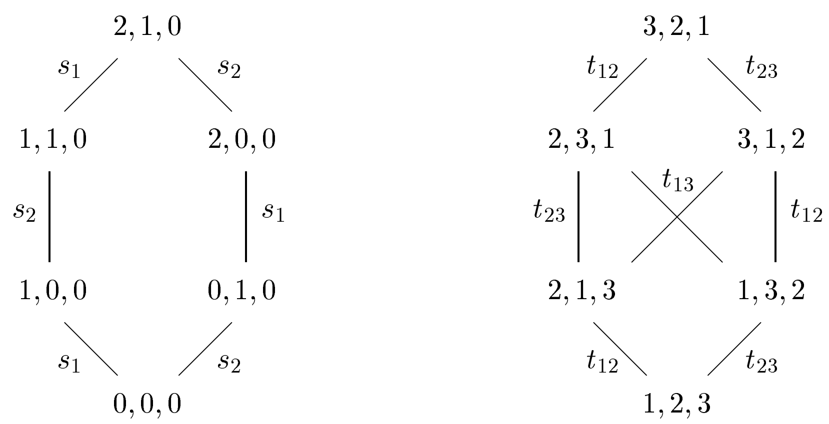


Figure 4.1: *Permutoèdre (des codes) et Bruhatoèdre de* $\mathfrak{S}_3$.

Les relations (4.4) et (4.5) impliquent que pour chaque permutation $\mu \in \mathfrak{S}_n$, il existe une différence divisée ∂_μ (le cas de Newton correspondant aux transpositions élémentaires). Celle-ci se définit, de manière non ambiguë grâce aux relations de tresse, en choisissant une décomposition réduite $s_{i_1} \ldots s_{i_k}$ de μ et en posant :

$$\partial_\mu := \partial_{i_1} \ldots \partial_{i_k} \quad . \tag{4.6}$$

La *formule de Leibnitz* exprime la différence divisée ∂_i d'un produit pq de deux polynômes :

$$\partial_i(pq) = (\partial_i p)q + (s_i p)\partial_i q \quad , \tag{4.7}$$

Cette formule peut se généraliser à n'importe quel opérateur ∂_μ. A la place de (4.7), nous écrivons :

$$\partial_i(pq) = \begin{array}{|c|c|}\hline \partial_i & p \\ \hline 1 & q \\ \hline\end{array} + \begin{array}{|c|c|}\hline s_i & p \\ \hline \partial_i & q \\ \hline\end{array} \quad , \tag{4.8}$$

où $\begin{array}{|c|c|}\hline \partial_i & p \\ \hline 1 & q \\ \hline\end{array} := (\partial_i p)q$ et $\begin{array}{|c|c|}\hline s_i & p \\ \hline \partial_i & q \\ \hline\end{array} := (s_i p)(\partial_i q)$. Il est alors clair que :

$$\partial_j\partial_i(pq) = \begin{array}{|c|c|}\hline \partial_j|\partial_i & p \\ \hline 1\,|\,1 & q \\ \hline\end{array} + \begin{array}{|c|c|}\hline s_j|\partial_i & p \\ \hline \partial_j|\,1 & q \\ \hline\end{array} + \begin{array}{|c|c|}\hline \partial_j|s_i & p \\ \hline 1\,|\partial_i & q \\ \hline\end{array} + \begin{array}{|c|c|}\hline s_j|s_i & p \\ \hline \partial_j|\partial_i & q \\ \hline\end{array} \quad , \tag{4.9}$$

en lisant chaque tableau comme le produit de ses lignes. Plus généralement, étant donnée une décomposition réduite $s_{i_1}\dots s_{i_k}$ d'une permutation μ, p et q deux polynômes, $\partial_\mu(pq)$ est représenté par la somme suivante impliquant $2^{l(\mu)} = 2^k$ termes :

$$\partial_\mu(pq) := \sum_{j=1}^{k} \begin{array}{|c|c|}\hline a_1|\cdots|a_k & p \\ \hline b_1|\cdots|b_k & q \\ \hline\end{array} \quad , \quad \text{où} \quad \begin{array}{|c|}\hline a_m \\ \hline b_m \\ \hline\end{array} \quad \text{dénote soit} \quad \begin{array}{|c|}\hline \partial_{i_m} \\ \hline 1 \\ \hline\end{array} \quad \text{soit} \quad \begin{array}{|c|}\hline s_{i_m} \\ \hline \partial_{i_m} \\ \hline\end{array} \;. \tag{4.10}$$

En 1982, Lascoux et Schützenberger montrent avec l'introduction des *polynômes de Schubert* [4] comment utiliser les différences divisées comme outil combinatoire algébrique. Ils sont définis comme les images par différences divisées du monôme $x^\delta = x_1^{n-1}x_2^{n-2}\dots x_n^0$:

$$\mathbf{X}_\mu := \partial_{\mu^{-1}\omega}(x^\delta) \quad , \tag{4.11}$$

où $\omega = [n, n-1, \dots, 2, 1]$ dénote la permutation maximale de $\mathfrak{S}_n$. Ce sont des polynômes en n variables (la dernière n'apparaissant pas), indicés par les permutations $\mu \in \mathfrak{S}_n$. On a parfois besoin d'indicer les polynômes de Schubert par des suites $\mathbf{i} = (i_1, \dots, i_n) \subset \mathbb{N}^n$ en prenant les codes de μ plutôt que les permutations elles-mêmes. On note alors $\mathbf{Y_i}$ le polynôme de Schubert correspondant à la suite $\mathbf{i} \in \mathbb{N}^n$. Le polynôme de Schubert $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$ est homogène de degré $l(\mu)$ en les variables $x_1, \dots, x_{n-1}$. Il est symétrique en x_i, x_{i+1} si $\mu(i) < \mu(i+1)$ et dès lors que μ appartient au *sous-groupe de Young* $\mathfrak{S}_i \times \mathfrak{S}_j$ *i.e.* $\mu = \mu'\mu''$, $\mu' \in \mathfrak{S}_i \times \{id\}$, $\mu'' \in \{id\} \times \mathfrak{S}_j$, on a $\mathbf{X}_\mu = \mathbf{X}_{\mu'}\mathbf{X}_{\mu''}$. Si μ est une permutation dominante, alors $\mathbf{X}_\mu = x^{\lambda(\mu)}$ et si μ est grassmannienne alors $\mathbf{X}_\mu = s_{\lambda(\mu)}(x_1, \dots, x_r)$ où r est la position de l'unique descente de μ et $s_{\lambda(\mu)}$ désigne la fonction de Schur correspondant à la partition $\lambda(\mu)$. Par ailleurs, on retrouve les fonctions complètes et les fonctions élémentaires pour les codes spéciaux suivants :

$$\mathbf{Y}_{\underbrace{0\dots0i}_{r}} = s_i(x_1, \dots, x_r), \quad i > 0 \quad , \quad \text{et} \quad \mathbf{Y}_{\underbrace{0\dots0}_{r-k}\underbrace{1\dots1}_{k}} = s_{\underbrace{1\dots1}_{k}}(x_1, \dots, x_r) \quad ,$$

en particulier, $\mathbf{X}_{s_i} = x_1 + \cdots + x_i$, pour $i = 1, \dots, n-1$.

Les polynômes de Schubert $\mathbf{X}_\mu$, $\mu \in \mathfrak{S}_n$ forment une $\mathbb{Z}$-base du sous-espace de $\mathbb{Z}[\mathbf{x}]$ engendré par les monômes $x^\alpha, \alpha \subseteq (n-1, n-2, \dots, 0)$ et par conséquent les polynômes de Schubert $\mathbf{X}_\mu$, $\mu \in \mathfrak{S}_\infty$ forment une $\mathbb{Z}$-base de l'anneau des polynômes $\mathbb{Z}[x_1, x_2, \dots]$ en une infinité de variables. Plus important, ces polynômes fournissent une base, adaptée à certains problèmes de géométrie algébrique ou théorie des représentations, de l'anneau des polynômes vu comme module libre sur l'anneau des polynômes symétriques. Nous reviendrons sur cette propriété essentielle un peu plus loin.

Considérant un deuxième ensemble d'indéterminées $\mathbf{y} := \{y_1, \ldots, y_n\}$, le *polynôme de Schubert double maximal* $\mathbb{X}_\omega$ est défini par :

$$\mathbb{X}_\omega(\mathbf{x}, \mathbf{y}) := \prod_{i+j \le n} (x_i - y_j) \quad , \tag{4.12}$$

où ω dénote l'élément maximal de $\mathfrak{S}_n$. Plus généralement, pour toute permutation $\mu \in \mathfrak{S}_n$, le *polynôme de Schubert double* $\mathbb{X}_\mu$ est défini comme image par différence divisée de $\mathbb{X}_\omega(\mathbf{x}, \mathbf{y})$:

$$\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) := \partial_{\mu^{-1}\omega}\Big(\mathbb{X}_\omega(\mathbf{x}, \mathbf{y})\Big) \quad , \tag{4.13}$$

formule dans laquelle les différences divisées n'agissent que sur les variables $\mathbf{x}$. Ces polynômes appartiennent à $\mathbb{Z}[x_1, \ldots, x_{n-1}, y_1, \ldots, y_{n-1}]$. Le polynôme $\mathbb{X}_\mu, \mu \in \mathfrak{S}_n$ est homogène de degré $l(\mu)$ en les variables $x_1, \ldots, x_{n-1}, y_1, \ldots, y_{n-1}$. La spécialisation de tous les y_i à 0 redonne les polynômes de Schubert simples.

La combinatoire des polynômes de Schubert est étroitement liée à la combinatoire des différences divisées et des deux ordres sur le groupe symétrique. Nous présentons ci-dessous côte à côte les polynômes de Schubert simples et doubles de $\mathfrak{S}_3$:

$$\begin{tikzcd}[column sep=small]
 & x_1^2x_2 \arrow[dl, dash, "\partial_1"'] \arrow[dr, dash, "\partial_2"] & \\
x_1x_2 \arrow[d, dash, "\partial_2"'] & & x_1^2 \arrow[d, dash, "\partial_1"] \\
x_1 \arrow[dr, dash, "\partial_1"'] & & x_1+x_2 \arrow[dl, dash, "\partial_2"] \\
 & 1 &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
 & & \mathbb{X}_{3,2,1} & & \\
 & & (x_1-y_1)(x_1-y_2)(x_2-y_1) \arrow[dl, dash, "\partial_1"'] \arrow[dr, dash, "\partial_2"] & & \\
\mathbb{X}_{2,3,1} & (x_1-y_1)(x_2-y_1) \arrow[d, dash, "\partial_2"'] & & (x_1-y_1)(x_1-y_2) \arrow[d, dash, "\partial_1"] & \mathbb{X}_{3,1,2} \\
\mathbb{X}_{2,1,3} & x_1-y_1 \arrow[dr, dash, "\partial_1"'] & & (x_1-y_1)+(x_2-y_2) \arrow[dl, dash, "\partial_2"] & \mathbb{X}_{1,3,2} \\
 & & 1 & & \\
 & & \mathbb{X}_{1,2,3} & &
\end{tikzcd}$$

L'interprétation géométrique des polynômes de Schubert est la source de nombreuses identités non triviales au niveau polynomial. Les calculs algébriques sur les polynômes de Schubert mettent en œuvre des objets combinatoires telles que les permutations, les codes ou les diagrammes de Rothe, ainsi que des techniques d'algèbre linéaire, grâce à l'existence d'un produit scalaire canonique accompagné d'opérateurs auto-adjoints.

Références bibliographiques

[1] M. Demazure. Désingularisation des variétés de Schubert. *Annales E.N.S.*, 6: 163–172, (1974).

[2] N. I.Bernstein, M. I.Gelfand, and S.I. Gelfand. Schubert cells and cohomology of the spaces G/P. *Uspekhi.Mat.Nauk*, 28: 3–26, (1973).

[3] A. Lascoux. Classes de Chern des variétés de drapeaux. *C.R. Acad. Sci. Paris*, 295: 393–398, (1982).

[4] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

5

Contributions à la théorie des polynômes de Schubert

Les différences divisées sont l'outil universel.

— ***Alain Lascoux.***

La structure multiplicative de l'anneau $Sym[x_1, \ldots, x_n]$ des polynômes symétriques en $x_1, \ldots, x_n$ est déterminée par la formule de Pieri qui exprime dans la base Schur le produit $h_i s_\lambda$ (ou $e_i s_\lambda$), $i \geq 0$, λ étant une partition et où h_i (resp. e_i) dénote la i-ième fonction symétrique complète (resp. élémentaire).

Si l'on exprime maintenant des polynômes non nécessairement symétriques dans la base Schubert, la structure multiplicative repose sur la multiplication par une variable. Cela met en jeu une certaine combinatoire des permutations décrite par la *formule de Monk* [17] (voir [16, (4.15')]).

Proposition 5.1 (Monk)

$$x_r \mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu - \sum_\eta \mathbf{X}_\eta \quad , \tag{5.1}$$

où les sommes sont prises sur les permutations $\nu = \mu\, t_{ri}$, $i > r$, $l(\nu) = l(\mu) + 1$ *et* $\eta = \mu\, t_{ir}$, $i < r$, $l(\eta) = l(\mu) + 1$.

Preuve. La démonstration originale s'appuyait sur un raisonnement géométrique. Nous donnons ci-dessous l'idée générale de la preuve algébrique que l'on peut trouver dans [16]. Si p est un polynôme en $x_1, \ldots, x_n$, alors p s'écrit [16, (4.14)] $p = \sum_{\nu \in \mathfrak{S}_{[n]}} \partial_\nu(p)_{|_0} \mathbf{X}_\nu$, où $\mathfrak{S}_{[n]}$ dénote l'ensemble des permutations dont le code est de longueur inférieure ou égale à n et où $\partial_\nu(p)_{|_0}$ est le terme constant du polynôme $\partial_\nu(p)$. Si $p = \sum \alpha_i x_i$ est cette fois un polynôme homogène de degré 1 en $x_1, \ldots, x_n$, alors [16, (4.15)] $p\mathbf{X}_\mu = \sum_{\nu \in \mathfrak{S}_{[n]}} \partial_\nu(p\mathbf{X}_\mu)_{|_0} \mathbf{X}_\nu$, mais par la formule de Leibnitz [16, (2.13)], on a $\partial_\nu(p\mathbf{X}_\mu) = \nu(p)\partial_\nu(\mathbf{X}_\mu) + \sum(\alpha_i - \alpha_j)\partial_{\nu t_{ij}} \mathbf{X}_\mu$, avec $l(\nu t_{ij}) = l(\mu)$ et $i < j$. Par suite, $\partial_\nu(p\mathbf{X}_\mu)_{|_0} = (\alpha_i - \alpha_j)$ si $\mu = \nu t_{ij}$ et $p\mathbf{X}_\mu = \sum(\alpha_i - \alpha_j)\mathbf{X}_{\nu t_{ij}}$, le cas de la formule de Monk ($p = x_r$) se déduisant immédiatement. ∎

5.1 Formule de Monk double

La formule de Monk permet par itération d'exprimer n'importe quel polynôme de l'anneau $\mathbb{Z}[x_1, x_2, \ldots]$ dans la base des polynômes de Schubert simples indicés par des permutations de $\mathfrak{S}_\infty$. Dans la suite, nous appellerons polynômes de Schubert les polynômes de Schubert doubles. Puisque les polynômes de Schubert font intervenir un deuxième ensemble de paramètres, ils forment aussi une base de l'anneau des polynômes $\mathbb{K}[x_1, x_2, \ldots]$ où l'anneau $\mathbb{K}$ des coefficients contient les variables $\mathbf{y}$. Plus généralement, l'ensemble des polynômes de Schubert dont on spécialise le deuxième ensemble de paramètres forme une base de cet anneau. Certains problèmes géométriques [4] imposent de choisir les polynômes de Schubert comme base de l'espace des polynômes en $\mathbf{x}$ à coefficients en $\mathbf{y}$. Pour retrouver la structure multiplicative de cet espace, nous devons décrire les produits des polynômes de Schubert par les variables x_r. Ceci est donné par la formule suivante (due à Kohnert et Veigneau) qui étend la formule de Monk (5.1) :

Proposition 5.2 ([10], 4.1)

$$x_r \mathbb{X}_\mu = y_{\mu(r)} \mathbb{X}_\mu + \sum_\nu \mathbb{X}_\nu - \sum_\eta \mathbb{X}_\eta \quad , \tag{5.2}$$

où les sommes sont prises sur les permutations $\nu = \mu\, t_{rj}$, $j > r$, $l(\nu) = l(\mu) + 1$ et $\eta = \mu\, t_{jr}$, $j < r$, $l(\eta) = l(\mu) + 1$.

Preuve. Tout d'abord nous vérifions la proposition lorsque $\mu = w_0$ est la permutation maximale de $\mathfrak{S}_n$: $(x_r - y_{n+1-r})\mathbb{X}_{w_0}$ est égal au polynôme de Schubert $\mathbf{X}_\mu$ car $\mu = w_0 t_{r,n+1}$ est une permutation dominante [16, (6.14)]. Nous pouvons donc écrire cette identité comme suit :

$$x_r \mathbb{X}_{w_0} = y_{w_0(r)} \mathbb{X}_{w_0} + \mathbb{X}_{w_0 t_{r,n+1}} \quad ,$$

ce qui est exactement (5.2). Maintenant, le cas d'une permutation arbitraire μ s'obtient facilement par récurrence décroissante sur la longueur de μ en utilisant la formule de Leibnitz pour calculer $\partial_i(x_r \mathbb{X}_\mu)$, i étant tel que $l(\mu s_i) = l(\mu) + 1$. Le cas initial contient le terme $y_{w_0(r)} \mathbb{X}_{w_0}$ et ce produit donne le terme $y_{\mu(r)} \mathbb{X}_\mu$ dans la formule (5.2). ■

A titre indicatif, nous donnons une autre preuve de la formule de Monk double, utilisant cette fois la généralisation de la formule d'interpolation de Newton au cas de plusieurs variables.

Preuve. Par la formule d'interpolation de Newton, le produit $p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x}, \mathbf{y})$ où $\mu \in \mathfrak{S}_n$ s'écrit :

$$p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) = \sum_{\nu \in \mathfrak{S}_m} \partial_\nu \Big(p(\mathbf{x}) \mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) \Big)_{|\mathbf{x}=\mathbf{y}} \mathbb{X}_\nu(\mathbf{x}, \mathbf{y}) \quad , \tag{5.3}$$

où $\partial_\nu(p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}))_{|\mathbf{x}=\mathbf{y}}$ désigne le polynôme $\partial_\nu(p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}))$ dans lequel on spécialise les variables x_i aux y_i. Comme dans le cas de la preuve de la formule de Monk simple, si $p = \sum \alpha_i\, x_i$, on a par [16, (2.13)] :

$$\partial_\nu \Big(p(\mathbf{x}) \mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) \Big) = \nu(p(\mathbf{x})) \partial_\nu \Big(\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) \Big) + \sum_{i<j} (\alpha_i - \alpha_j) \partial_{\nu t_{ij}} \Big(\mathbb{X}_\mu(\mathbf{x}, \mathbf{y}) \Big) \quad ,$$

où $l(\nu t_{ij}) = l(\mu)$. Par suite, (5.3) devient :

$$\begin{aligned} p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x},\mathbf{y}) &= \sum_{\nu\in\mathfrak{S}_m} \nu\Big(p(\mathbf{x})\Big)_{|\mathbf{x}=\mathbf{y}} \partial_\nu\Big(\mathbb{X}_\mu(\mathbf{x},\mathbf{y})\Big)_{|\mathbf{x}=\mathbf{y}} \mathbb{X}_\nu(\mathbf{x},\mathbf{y}) \\ &+ \sum_{\nu\in\mathfrak{S}_m}\sum_{i<j}(\alpha_i-\alpha_j)\partial_{\nu t_{ij}}\Big(\mathbb{X}_\mu(\mathbf{x},\mathbf{y})\Big)_{|\mathbf{x}=\mathbf{y}} \mathbb{X}_\nu(\mathbf{x},\mathbf{y}) \quad . \end{aligned} \tag{5.4}$$

La spécialisation $\mathbf{x}=\mathbf{y}$ annule tous les polynômes de Schubert sauf le polynôme :

$$\mathbb{X}_{1,\dots,m}(\mathbf{x},\mathbf{y})_{|\mathbf{x}=\mathbf{y}} = 1 \quad .$$

Les seules permutations ν qui contribuent dans les deux sommes précédentes sont donc respectivement :

$$\begin{cases} \nu\Big(p(\mathbf{x})\Big)_{|\mathbf{x}=\mathbf{y}} \partial_\nu\Big(\mathbb{X}_\mu(\mathbf{x},\mathbf{y})\Big)_{|\mathbf{x}=\mathbf{y}} = \nu(p(\mathbf{x}))_{|\mathbf{x}=\mathbf{y}} & \to \quad \nu=\mu \quad , \\ \partial_{\nu t_{ij}}\Big(\mathbb{X}_\mu(\mathbf{x},\mathbf{y})\Big)_{|\mathbf{x}=\mathbf{y}} = 1 & \to \quad \nu=\mu t_{ij} \quad . \end{cases}$$

Par suite :

$$p(\mathbf{x})\mathbb{X}_\mu(\mathbf{x},\mathbf{y}) = \mu\Big(p(\mathbf{y})\Big)\ \mathbb{X}_\mu(\mathbf{x},\mathbf{y}) + \sum_{i<j}(\alpha_i-\alpha_j)\ \mathbb{X}_{\mu t_{ij}}(\mathbf{x},\mathbf{y}) \quad ,$$

avec toujours $l(\mu t_{ij}) = l(\mu)+1$. En particulier, pour $p = x_r$, on retrouve la formule de Monk double (5.2). ■

La formule de Monk (5.1) correspond à la spécialisation de toutes les variables y_i à 0 dans (5.2). Sa version pour les polynômes de Schubert doubles permet par itération d'exprimer n'importe quel polynôme en $\mathbf{x}$ dans la base des polynômes de Schubert à coefficients polynomiaux en $\mathbf{y}$. Nous donnons ci-dessous une application de cette formule en géométrie.

5.2 Application de la formule de Monk double

Les fonctions de Schur apparaissent en géométrie algébrique comme des classes de cohomologie de certaines variétés déterminantales. Une situation plus générale survient lorsque l'on choisit un fibré vectoriel V et deux drapeaux de fibrés vectoriels : $A_1 \hookrightarrow A_2 \hookrightarrow \cdots \hookrightarrow A_n = V$ et $V = B_n \twoheadrightarrow B_{n-1} \twoheadrightarrow \cdots \twoheadrightarrow B_1$. Les classes de variétés déterminantales sont alors des polynômes en les classes de Chern $\{x_1,\dots,x_n\},\{y_1,\dots,y_n\}$ des fibrés en droites associés aux deux drapeaux et on a les conditions que $f(\mathbf{x}) = f(\mathbf{y})$ pour toutes les fonctions symétriques f, car dans ce cas f est une fonction de V uniquement [4]. Nous allons montrer qu'après identification des fonctions symétriques en $\mathbf{x}$ et des fonctions symétriques en $\mathbf{y}$, c'est-à-dire modulo l'idéal $\mathcal{I}(\mathbf{x},\mathbf{y})$ engendré par les $e_k(\mathbf{x}) - e_k(\mathbf{y})$, il n'y a pas de terme $\mathbb{X}_\nu, \nu \in \mathfrak{S}_{n+1}\backslash\mathfrak{S}_n$ dans le produit $x_r\mathbb{X}_\mu, \mu\in\mathfrak{S}_n$. Ainsi, nous fixons désormais l'anneau $\mathbb{K}$ des coefficients à :

$$\mathbb{K} := \frac{Sym[x_1,\dots,x_n]\otimes\mathbb{Z}[y_1,\dots,y_n]}{\mathcal{I}(\mathbf{x},\mathbf{y})} \quad .$$

L'anneau $\mathbb{Z}[\mathbf{y}][\mathbf{x}]\big/\mathcal{I}(\mathbf{x},\mathbf{y})$ est un module libre de dimension $n!$ sur l'anneau $\mathbb{K}$, avec comme base distinguée les $\mathbf{X}_\mu$ ou les $\mathbb{X}_\mu, \mu \in \mathfrak{S}_n$. Nous notons cet anneau $\mathbb{K}[\mathbf{x}]$. Dans cet anneau, on a la propriété suivante, due à Kohnert et Veigneau :

Proposition 5.3 ([10], 4.2) *Soit $\mu \in \mathfrak{S}_n$ et $1 \leq r \leq n$. Dans l'anneau $\mathbb{K}[\mathbf{x}]$, on a :*

$$x_r \mathbb{X}_\mu \equiv y_{\mu(r)} \mathbb{X}_\mu + \sum_\nu \mathbb{X}_\nu - \sum_\eta \mathbb{X}_\eta \quad , \tag{5.5}$$

où les sommes sont prises sur les permutations de $\mathfrak{S}_n$, $\nu = \mu\, t_{rj}$, $j > r$, $l(\nu) = l(\mu) + 1$ et $\eta = \mu\, t_{jr}$, $j < r$, $l(\eta) = l(\mu) + 1$.

Preuve. Dans la formule (5.2), il y a au plus un terme impliquant une permutation de $\mathfrak{S}_{n+1}$. Ce terme provient du cas $\mu = w_0$. Par conséquent, la preuve de la proposition se réduit à vérifier l'identité suivante :

$$\forall r,\ 1 \leq r \leq n, \qquad (x_r - y_{n+1-r})\mathbb{X}_{w_0} \equiv 0 \quad .$$

En fait, nous allons montrer une identité plus précise, c'est-à-dire qu'un certain facteur de l'expression précédente s'annule.

Lemme 5.4 ([10], 4.3) *Soit un entier r tel que $1 \leq r \leq n$. Alors, modulo l'idéal $\mathcal{I}(\mathbf{x},\mathbf{y})$, on a :*

$$\prod_{1\leq i\leq r,\, 1\leq j\leq n+1-r} (x_i - y_j) \equiv 0 \quad . \tag{5.6}$$

Preuve. Etant donnés deux alphabets A et B de cardinal respectivement l et m, alors le produit $R(A,B) := \prod_{a\in A,\, b\in B}(a-b)$ est un certain polynôme de Schubert vexillaire pour les deux alphabets A et B. Puisqu'il est vexillaire, il a aussi une expression déterminantale [16, page 94] :

$$R(A,B) = \left| h_{m+j-i}(A-B) \right|_{1\leq i,j\leq l} \quad ,$$

où par définition les $h_k(A-B)$ sont les coefficients de la fonction rationnelle en z :

$$f(A,B) := \prod_{b\in B}(1-zb)\Big/\prod_{a\in A}(1-za) \quad .$$

Cette fonction est par suite égale à $R(\{x_1,\ldots,x_r\},\{y_1,\ldots,y_{n+1-r}\})$ qui est un déterminant en les $h_k(\{x_1,\ldots,x_r\} - \{y_1,\ldots,y_{n+1-r}\})$. Considérons maintenant une variable supplémentaire u. Le produit $g := \prod_{1\leq i\leq n}(1-zx_i)(1-zu)$ est égal à $\prod_{1\leq j\leq n}(1-zy_j)(1-zu)$ car les fonctions symétriques élémentaires en $x_1,\ldots,x_n$ sont égales aux fonctions symétriques élémentaires en $y_1,\ldots,y_n$. Ainsi, en multipliant le numérateur et le dénominateur de $f(\{x_1,\ldots,x_r\},\{y_1,\ldots,y_{n+1-r}\})$ par g, on transforme f en la fonction rationnelle :

$$f(\{y_{n+2-r},\ldots,y_n,u\},\{x_{r+1},\ldots,x_n,u\}) \quad ,$$

sans changer sa valeur. Cela implique que $h_k(\{x_1,\ldots,x_r\} - \{y_1,\ldots,y_{n+1-r}\})$ doit coïncider avec $h_k(\{y_{n+2-r},\ldots,y_n,u\} - \{x_{r+1},\ldots,x_n,u\})$. Cette transformation montre que le déterminant est égal à $R(\{y_{n+2-r},\ldots,y_n,u\},\{x_{r+1},\ldots,x_n,u\})$, qui s'annule à cause du facteur $(u-u)$. ■

La preuve de la proposition 5.3 s'en suit immédiatement. ■

Exemple 5.5 On se place dans $\mathfrak{S}_3$ et on considère le produit $(x_2 - y_2)\mathbb{X}_{w_0}$ qui est égal à $R(\{x_1, x_2\}, \{y_1, y_2\})$. Modulo l'idéal $\mathcal{I}(\{x_1, x_2, x_3\}, \{y_1, y_2, y_3\})$, on a :

$$\begin{aligned}
R(\{x_1,x_2\},\{y_1,y_2\}) &= \begin{vmatrix} h_2(x_1+x_2-y_1-y_2) & h_3(x_1+x_2-y_1-y_2) \\ h_1(x_1+x_2-y_1-y_2) & h_2(x_1+x_2-y_1-y_2) \end{vmatrix} \\
&= \begin{vmatrix} h_2(y_3-x_3) & h_3(y_3-x_3) \\ h_1(y_3-x_3) & h_2(y_3-x_3) \end{vmatrix} = \begin{vmatrix} y_3\,(y_3-x_3) & y_3^2\,(y_3-x_3) \\ (y_3-x_3) & y_3\,(y_3-x_3) \end{vmatrix} \quad ,
\end{aligned}$$

qui s'annule clairement.

La formule de Monk double donne donc la structure multiplicative de l'anneau des polynômes en $x_1, \ldots, x_n$ avec comme base les polynômes de Schubert (doubles) modulo identification des fonctions symétriques en $\mathbf{x}$ et des fonctions symétriques en $\mathbf{y}$.

5.3 Formule de Pieri double

Comme nous l'avons déjà précisé, $\mathbf{X}_{s_r} = x_1 + \cdots + x_r$. De fait, la formule de Monk s'énonce aussi de manière équivalente par $\mathbf{X}_{s_r}\,\mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu$, où la somme est prise sur toutes les permutations $\nu = \mu\, t_{ij}$, $i \leq r < j$, $l(\nu) = l(\mu) + 1$. La permutation s_r a pour code $[0 \ldots 010 \ldots 0]$ et l'on peut écrire la formule précédente :

$$\mathbf{Y}_{0\ldots010\ldots0}\,\mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu \quad . \tag{5.7}$$

Nous avons par ailleurs vu dans le chapitre précédent que les polynômes de Schubert simples indicés par les codes $[0 \ldots 0k0 \ldots 0]$ (respectivement $[0 \ldots 01 \ldots 10 \ldots 0]$) sont des fonctions symétriques complètes (resp. élémentaires). Nous avons précisé au début de ce chapitre que la structure multiplicative de $Sym[x_1, \ldots, x_n]$ nécessite la connaissance des produits $h_i s_\lambda$ ou $e_i s_\lambda$. Ces produits sont donnés par la formule de Pieri qui a de nombreuses interprétations en géométrie et en théorie des représentations. La propriété essentielle de cette formule est l'absence de multiplicité. Nous allons voir qu'il en est de même dans le cas des polynômes de Schubert où l'on a les formules suivantes :

$$\mathbf{Y}_{0\ldots0\,k\,0\ldots0}\,\mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu \quad , \qquad \text{et} \qquad \mathbf{Y}_{0\ldots0\,1\ldots1\,0\ldots0}\,\mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu \quad ,$$

qui sont dites par analogie avec la théorie des fonctions symétriques, *formules de Pieri* [12].

Revenons tout d'abord à la formule de Pieri classique donnant le développement $h_j s_\lambda$ dans la base des fonctions de Schur. L'opération élémentaire est l'ajout de $i \leq j$ boîtes adjacentes dans une ligne de λ. On visualise mieux cette opération sur le diagramme de Ferrers de λ comme le montre la figure 5.1 :

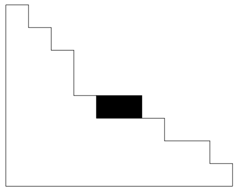


Figure 5.1: *Opération élémentaire d'ajout de i boîtes adjacentes dans la formule de Pieri $h_j s_\lambda, j \geq i$.*

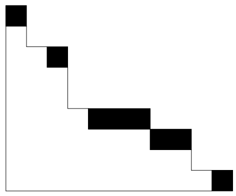


Figure 5.2: *Formule de Pieri $h_j s_\lambda$.*

La formule Pieri qui donne les fonctions de Schur apparaissant dans le développement du produit $h_j s_\lambda$ est alors une succession de ces opérations élémentaires (voir figure 5.2). De manière condensée, on peut considérer simultanément le produit de s_λ par tous les h_i :

$$\Big(\sum_{i=0}^{\infty} h_i\Big) s_\lambda \quad . \tag{5.8}$$

Il est clair que les opérations (élémentaires) se font sur les lignes de la partition λ de manière indépendante. Par exemple :

$$\Big(\sum_{i=0}^{\infty} h_i\Big) s_{5,2} = \sum_{5 \leq i_1} \sum_{2 \leq i_2 \leq 5} \sum_{0 \leq i_3 \leq 2} s_{i_1,i_2,i_3} \quad .$$

Au passage, il faut remarquer la dualité qui existe entre fonctions symétriques élémentaires et fonctions symétriques complètes au niveau de la formule de Pieri. La multiplication des fonctions de Schur par les fonctions élémentaires s'obtient en remplaçant les diagrammes par leurs conjugués dans la description ci-dessus.

Pour les polynômes de Schubert, l'étape élémentaire dans la multiplication par une fonction complète est la multiplication par un cycle appelé (k, p)-*soulèvement droit* dans [12]. Plus précisément, soient p et k deux entiers positifs et $\mu \in \mathfrak{S}_n$. Alors, un (k, p)-soulèvement droit de μ est un cycle ζ tel que :

① μ et $\mu\zeta = \nu$ diffèrent en $k+1$ positions $i_0, \ldots, i_k$, avec $i_0 \leq p$, $p+1 \leq i_1 \leq \cdots \leq i_k$;

② les codes des inversions des sous mots $\mu_{i_0} \ldots \mu_{i_k}$ et $\nu_{i_0} \ldots \nu_{i_k}$ sont respectivement $[0, c_1, \ldots, c_k]$ et $[k, c_1, \ldots, c_k]$;

③ $l(\mu\zeta) = l(\mu) + k$.

De manière plus imagée, nous appellerons k le degré et p la barrière. Graphiquement, un (k,p)-soulèvement droit se visualise sur la permutation μ. Dans la figure suivante, nous avons représenté les 15 entrées de la permutation $\mu \in \mathfrak{S}_{15}$ par des points. Ces points sont noirs lorsque les entrées correspondantes interviennent dans le $(4,7)$-soulèvement droit suivant :

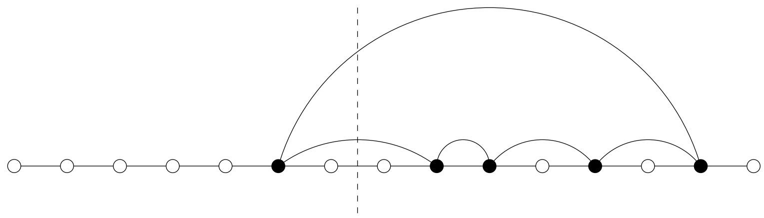
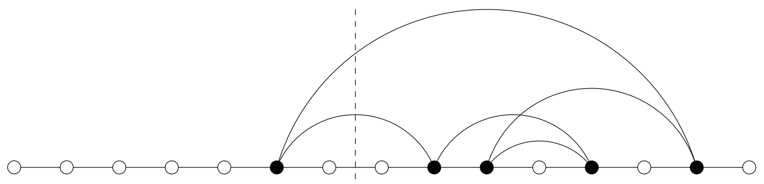


Figure 5.3: *Exemple d'un* $(4,7)$*-soulèvement droit.*

Cette figure montre bien que le $(4,7)$-soulèvement droit ne fait intervenir qu'une seule valeur à gauche de la barrière (entre les positions 7 et 8) contre $k=4$ à droite. La figure 5.4 donne un exemple plus concret de cette situation :



4	14	1	6	8	2	11	12	3	7	15	5	9	10	13
					0			0	1		0		0	
					4			0	1		0		0	
4	14	1	6	8	10	11	12	2	5	15	3	9	7	13

Figure 5.4: $(4,7)$*-soulèvement droit pour la permutation de* $\mathfrak{S}_{15}$ $\mu = [4,14,1,6,8,2,11,12,3,7,15,5,9,10,13]$. *Figurent de haut en bas la permutation* μ*, le code restreint aux positions de* μ *modifiées par le cycle* ζ*, le code restreint de* $\mu\zeta$ *et enfin la permutation* $\mu\zeta$*.*

On définit de manière symétrique un (k,p)-soulèvement gauche. Chacun des termes dans la formule de Pieri développant le produit $h_i(x_1,\ldots,x_p)\mathbf{X}_\mu$ s'obtient comme un produit de (k,p)-soulèvements droits, la somme des poids $\sum k$ étant égale à i. On a une formule analogue pour $e_i(x_1,\ldots,x_p)\mathbf{X}_\mu$ en remplaçant les soulèvements droits par des soulèvements gauches.

La condition que les lignes (resp. colonnes) n'interfèrent pas dans le cas des diagrammes de Ferrers est ici remplacée par une condition d'accroissement de longueur dans le calcul des produits $\left(\sum_{i=0}^{\infty} h_i(x_1,\ldots,x_p)\right)\mathbf{X}_\mu$ et $\left(\sum_{i=0}^{\infty} e_i(x_1,\ldots,x_p)\right)\mathbf{X}_\mu$.

Proposition 5.6 ([12]) *Soit* $\mu \in \mathfrak{S}_n$ *et* $f = h_m(x_1, \ldots, x_p)$ *(resp.* $f = e_m(x_1, \ldots, x_p)$*). Alors :*

$$f\,\mathbf{X}_\mu = \sum_\nu \mathbf{X}_\nu \quad , \tag{5.9}$$

où la somme est étendue à toutes les permutations ν *obtenues à partir de* μ *par des produits de* (i,p)*-soulèvements droits (resp. gauches)* ζ_i *tels que* $l(\nu) - l(\mu) = \sum l(\zeta_i)$*.*

Preuve. La preuve se fait par récurrence au moyen de l'identité sur les fonctions complètes :

$$h_m(x_1, \ldots, x_p) = h_m(x_1, \ldots, x_{p-1}) + x_p h_{m-1}(x_1, \ldots, x_p) \quad ,$$

respectivement sur les fonctions élémentaires :

$$e_m(x_1, \ldots, x_p) = e_m(x_1, \ldots, x_{p-1}) + x_p e_{m-1}(x_1, \ldots, x_{p-1}) \quad .$$

Une preuve géométrique a été obtenue par Sottile [18]. ∎

De manière similaire, la formule (5.9) peut être donnée dans le cas des polynômes de Schubert (doubles). Nous rappelons que la fonction symétrique complète $h_m(x_1, \ldots, x_p)$ (resp. $e_m(x_1, \ldots, x_p)$) est égale au polynôme de Schubert simple $\mathbf{Y}_{0^{p-1}m}$ (resp. $\mathbf{Y}_{0^{p-m}1^m}$). Par ailleurs, nous notons $\mathbf{Y}_{0^{p-1}m}(\mu \to \nu)$ l'image de la fonction symétrique $\mathbf{Y}_{0^{p-1}m}$ par la spécialisation des x_i en les y_j où j parcourt l'ensemble des valeurs à gauche de la barrière en position p ou déplacées par le cycle ζ permettant de passer de μ à ν. Dans [15], nous avons obtenu avec Lascoux la propriété suivante :

Proposition 5.7 *Soit* μ *une permutation de* $\mathfrak{S}_n$ *et soient* m, p *deux entiers positifs. Alors :*

$$\mathbf{Y}_{0^{p-1}m}\,\mathbb{X}_\mu = \sum_\nu \mathbf{Y}_{0^{p-1+i}m-i}(\mu \to \nu)\,\mathbb{X}_\nu \quad , \tag{5.10}$$

où la somme est étendue à toutes les permutations ν *telles que la décomposition en cycles de* $\nu\mu^{-1}$ *ne comporte que des* (k,p)*-soulèvements droits avec* $l(\nu) \leq l(\mu) + m$ *et où* $i = l(\nu) - l(\mu)$*.*

Preuve. A nouveau, la preuve repose sur l'étude des différents termes obtenus par récurrence sur les fonctions complètes :

$$\mathbf{Y}_{0^{p-1}m} = \mathbf{Y}_{0^{p-2}m} + x_p \mathbf{Y}_{0^{p-1}m-1} \quad .$$

Cette récurrence indique que les polynômes de Schubert qui apparaissent dans le produit $\mathbf{Y}_{0^{p-1}m}\,\mathbb{X}_\mu$ proviennent soit de $\mathbf{Y}_{0^{p-2}m}\,\mathbb{X}_\mu$, soit de $\mathbf{Y}_{0^{p-1}m-1}\,\mathbb{X}_\mu$ suivi d'une multiplication par x_p (formule de Monk double (5.2)), soit de la somme de deux termes provenant de ces deux produits. ∎

Exemple 5.8 On considère la permutation $\mu = [7, 2, 8, 4, 3, 9, 1, 6, 5]$ et le polynôme de Schubert $\mathbb{X}_\mu$. Les termes du produit de $\mathbb{X}_\mu$ par $h_3(x_1, x_2, x_3, x_4)$ proviennent des termes de $h_3(x_1, x_2, x_3)\mathbb{X}_\mu$ et $x_4 h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ puisque :

$$h_3(x_1, x_2, x_3, x_4) = h_3(x_1, x_2, x_3) + x_4 h_2(x_1, x_2, x_3, x_4) \quad .$$

Plus précisément, la table suivante présente les résultats des produits $h_3(x_1, x_2, x_3)\mathbb{X}_\mu$ et $h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ qui sont tous deux obtenus par la formule de Pieri double (5.10). Nous indiquons pour chaque polynôme de Schubert de la première colonne son coefficient dans ces deux produits :

Polynôme de Schubert	**Coefficient dans** $h_3\mathbb{X}_\mu$	**Coefficient dans** $h_2\mathbb{X}_\mu$
$\mathbb{X}_{7,2,9,4,3,8,1,6,5,10,11}$	$h_2(y_2, y_7, y_8, y_9)$	$h_1(y_2, y_4, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,8,4,3,9,1,6,5,10,11}$	$h_3(y_2, y_7, y_8)$	$h_2(y_2, y_4, y_7, y_8)$
$\mathbb{X}_{7,2,10,4,3,8,1,6,5,9,11}$	$h_1(y_2, y_7, y_8, y_9, y_{10})$	$h_0(y_2, y_4, y_7, y_8, y_9, y_{10})$
$\mathbb{X}_{7,2,11,4,3,8,1,6,5,9,10}$	$h_0(y_2, y_7, y_8, y_{11})$	
$\mathbb{X}_{7,4,10,2,3,8,1,6,5,9,11}$	$h_0(y_2, y_4, y_7, y_8, y_{10})$	
$\mathbb{X}_{7,3,10,4,2,8,1,6,5,9,11}$	$h_0(y_2, y_3, y_7, y_8, y_{10})$	
$\mathbb{X}_{7,4,9,2,3,8,1,6,5,10,11}$	$h_1(y_2, y_4, y_7, y_8, y_9)$	
$\mathbb{X}_{7,3,9,4,2,8,1,6,5,10,11}$	$h_1(y_2, y_3, y_7, y_8, y_9)$	$h_0(y_2, y_3, y_4, y_7, y_8, y_9)$
$\mathbb{X}_{7,4,8,2,3,9,1,6,5,10,11}$	$h_2(y_2, y_4, y_7, y_8)$	
$\mathbb{X}_{7,3,8,4,2,9,1,6,5,10,11}$	$h_2(y_2, y_3, y_7, y_8)$	$h_1(y_2, y_3, y_4, y_7, y_8)$
$\mathbb{X}_{7,6,8,2,3,9,1,5,4,10,11}$	$h_0(y_2, y_6, y_7, y_8)$	
$\mathbb{X}_{7,5,8,3,2,9,1,6,4,10,11}$	$h_0(y_2, y_5, y_7, y_8)$	
$\mathbb{X}_{7,6,8,3,2,9,1,4,5,10,11}$	$h_0(y_2, y_6, y_7, y_8)$	
$\mathbb{X}_{7,6,9,2,3,8,1,4,5,10,11}$	$h_0(y_2, y_6, y_7, y_8, y_9)$	
$\mathbb{X}_{7,5,9,2,3,8,1,6,4,10,11}$	$h_0(y_2, y_5, y_7, y_8, y_9)$	
$\mathbb{X}_{7,4,9,3,2,8,1,6,5,10,11}$	$h_0(y_2, y_4, y_8, y_9)$	
$\mathbb{X}_{7,6,8,2,3,9,1,4,5,10,11}$	$h_1(y_2, y_4, y_6, y_7, y_8)$	
$\mathbb{X}_{7,5,8,2,3,9,1,6,4,10,11}$	$h_1(y_2, y_4, y_5, y_7, y_8)$	
$\mathbb{X}_{7,4,8,3,2,9,1,6,5,10,11}$	$h_1(y_2, y_3, y_4, y_7, y_8)$	
$\mathbb{X}_{7,2,8,9,3,6,1,4,5,10,11}$		$h_0(y_2, y_4, y_6, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,8,9,3,5,1,6,4,10,11}$		$h_0(y_2, y_4, y_5, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,8,6,3,9,1,5,4,10,11}$		$h_0(y_2, y_4, y_5, y_6, y_7, y_8)$
$\mathbb{X}_{7,2,9,6,3,8,1,4,5,10,11}$		$h_0(y_2, y_4, y_6, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,9,5,3,8,1,6,4,10,11}$		$h_0(y_2, y_4, y_5, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,8,9,3,4,1,6,5,10,11}$		$h_1(y_2, y_4, y_7, y_8, y_9)$
$\mathbb{X}_{7,2,8,6,3,9,1,4,5,10,11}$		$h_1(y_2, y_4, y_6, y_7, y_8)$
$\mathbb{X}_{7,2,8,5,3,9,1,6,4,10,11}$		$h_1(y_2, y_4, y_5, y_7, y_8)$
$\mathbb{X}_{7,2,8,10,3,4,1,6,5,9,11}$		$h_0(y_2, y_4, y_7, y_8, y_9, y_{10})$
$\mathbb{X}_{7,3,8,9,2,4,1,6,5,10,11}$		$h_0(y_2, y_3, y_4, y_7, y_8, y_9)$
$\mathbb{X}_{7,3,8,6,2,9,1,4,5,10,11}$		$h_0(y_2, y_3, y_4, y_6, y_7, y_8)$
$\mathbb{X}_{7,3,8,5,2,9,1,6,4,10,11}$		$h_0(y_2, y_3, y_4, y_5, y_7, y_8)$

Puisque ce sont les termes de $x_4h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ qui nous intéressent et non ceux de $h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$, la troisième colonne doit être multipliée par x_4. Plus exactement, chaque polynôme de Schubert $\mathbb{X}_\nu$ de la première colonne intervenant dans le développement de $h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ doit être multiplié par x_4. Chaque multiplication, qui correspond à la formule de Monk double, produit des termes qui pourront soit contribuer directement au résultat, soit s'annuler avec un terme de la deuxième colonne, soit enfin se lier à un terme de la deuxième colonne pour produire un seul terme. Dans ce dernier cas, la situation est la suivante : le polynôme de Schubert $\mathbb{X}_\nu$ a dans le développement de $h_3(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ un coefficient égal à :

$$h_i(\mathbf{y}) + y_j h_{i-1}(\mathbf{y}, y_j) \quad ,$$

où $\mathbf{y}$ est un sous-ensemble de variables de $\{y_1, \dots, y_{11}\}$ et $y_j \not\in \mathbf{y}$. Mais puisque :

$$h_i(\mathbf{y}) + y_j h_{i-1}(\mathbf{y}, y_j) = h_i(\mathbf{y}, y_j) \quad ,$$

les deux termes contribueront pour un seul terme dans le résultat final. La table suivante donne pour chacun des polynômes de Schubert $\mathbb{X}_\nu$ apparaissant dans le développement de $h_2(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$, le produit $x_4\mathbb{X}_\nu$:

$\mathbb{X}_\nu$	$x_4\mathbb{X}_\nu$
$\mathbb{X}_{7,2,8,9,3,4,1,6,5,10,11}$	$-\mathbb{X}_{7,2,9,8,3,4,1,6,5,10,11} + \mathbb{X}_{7,2,8,10,3,4,1,6,5,9,11} + y_9\mathbb{X}_{7,2,8,9,3,4,1,6,5,10,11}$
$\mathbb{X}_{7,3,8,9,2,4,1,6,5,10,11}$	$-\mathbb{X}_{7,3,9,8,2,4,1,6,5,10,11} + \mathbb{X}_{7,3,8,10,2,4,1,6,5,9,11} + y_9\mathbb{X}_{7,3,8,9,2,4,1,6,5,10,11}$
$\mathbb{X}_{7,2,8,6,3,9,1,5,4,10,11}$	$-\mathbb{X}_{7,6,8,2,3,9,1,5,4,10,11} + \mathbb{X}_{7,2,8,9,3,6,1,5,4,10,11} + y_6\mathbb{X}_{7,2,8,6,3,9,1,5,4,10,11}$
$\mathbb{X}_{7,2,8,6,3,9,1,4,5,10,11}$	$-\mathbb{X}_{7,6,8,2,3,9,1,4,5,10,11} + \mathbb{X}_{7,2,8,9,3,6,1,4,5,10,11} + y_6\mathbb{X}_{7,2,8,6,3,9,1,4,5,10,11}$
$\mathbb{X}_{7,2,8,9,3,5,1,6,4,10,11}$	$-\mathbb{X}_{7,2,9,8,3,5,1,6,4,10,11} + \mathbb{X}_{7,2,8,10,3,5,1,6,4,9,11} + y_9\mathbb{X}_{7,2,8,9,3,5,1,6,4,10,11}$
$\mathbb{X}_{7,2,8,4,3,9,1,6,5,10,11}$	$-\mathbb{X}_{7,4,8,2,3,9,1,6,5,10,11} + \mathbb{X}_{7,2,8,9,3,4,1,6,5,10,11} + \mathbb{X}_{7,2,8,6,3,9,1,4,5,10,11}$ $+\mathbb{X}_{7,2,8,5,3,9,1,6,4,10,11} + y_4\mathbb{X}_{7,2,8,4,3,9,1,6,5,10,11}$
$\mathbb{X}_{7,2,9,4,3,8,1,6,5,10,11}$	$-\mathbb{X}_{7,4,9,2,3,8,1,6,5,10,11} + \mathbb{X}_{7,2,9,8,3,4,1,6,5,10,11} + \mathbb{X}_{7,2,9,6,3,8,1,4,5,10,11}$ $+ \mathbb{X}_{7,2,9,5,3,8,1,6,4,10,11} + y_4\mathbb{X}_{7,2,9,4,3,8,1,6,5,10,11}$
$\mathbb{X}_{7,3,9,4,2,8,1,6,5,10,11}$	$-\mathbb{X}_{7,4,9,3,2,8,1,6,5,10,11} + \mathbb{X}_{7,3,9,8,2,4,1,6,5,10,11} + \mathbb{X}_{7,3,9,6,2,8,1,4,5,10,11}$ $+ \mathbb{X}_{7,3,9,5,2,8,1,6,4,10,11} + y_4\mathbb{X}_{7,3,9,4,2,8,1,6,5,10,11}$
$\mathbb{X}_{7,3,8,6,2,9,1,4,5,10,11}$	$-\mathbb{X}_{7,6,8,3,2,9,1,4,5,10,11} + \mathbb{X}_{7,3,8,9,2,6,1,4,5,10,11} + y_6\mathbb{X}_{7,3,8,6,2,9,1,4,5,10,11}$
$\mathbb{X}_{7,2,8,10,3,4,1,6,5,9,11}$	$-\mathbb{X}_{7,2,10,8,3,4,1,6,5,9,10} + \mathbb{X}_{7,2,8,11,3,4,1,6,5,9,10} + y_{10}\mathbb{X}_{7,2,8,10,3,4,1,6,5,9,11}$
$\mathbb{X}_{7,2,8,5,3,9,1,6,4,10,11}$	$-\mathbb{X}_{7,5,8,2,3,9,1,6,4,10,11} + \mathbb{X}_{7,2,8,9,3,5,1,6,4,10,11} + \mathbb{X}_{7,2,8,6,3,9,1,5,4,10,11}$ $+ y_5\mathbb{X}_{7,2,8,5,3,9,1,6,4,10,11}$
$\mathbb{X}_{7,3,8,4,2,9,1,6,5,10,11}$	$-\mathbb{X}_{7,4,8,3,2,9,1,6,5,10,11} + \mathbb{X}_{7,3,8,9,2,4,1,6,5,10,11} + \mathbb{X}_{7,3,8,6,2,9,1,4,5,10,11}$ $+ \mathbb{X}_{7,3,8,5,2,9,1,6,4,10,11} + y_4\mathbb{X}_{7,3,8,4,2,9,1,6,5,10,11}$
$\mathbb{X}_{7,2,10,4,3,8,1,6,5,9,11}$	$-\mathbb{X}_{7,4,10,2,3,8,1,6,5,9,11} + \mathbb{X}_{7,2,10,8,3,4,1,6,5,9,11} + \mathbb{X}_{7,2,10,6,3,8,1,4,5,9,11}$ $+ \mathbb{X}_{7,2,10,5,3,8,1,6,4,9,11} + y_4\mathbb{X}_{7,2,10,4,3,8,1,6,5,9,11}$
$\mathbb{X}_{7,3,8,5,2,9,1,6,4,10,11}$	$-\mathbb{X}_{7,5,8,3,2,9,1,6,4,10,11} + \mathbb{X}_{7,3,8,9,2,5,1,6,4,10,11} + \mathbb{X}_{7,3,8,6,2,9,1,5,4,10,11}$ $+ y_5\mathbb{X}_{7,3,8,5,2,9,1,6,4,10,11}$
$\mathbb{X}_{7,2,9,5,3,8,1,6,4,10,11}$	$-\mathbb{X}_{7,5,9,2,3,8,1,6,4,10,11} + \mathbb{X}_{7,2,9,8,3,5,1,6,4,10,11} + \mathbb{X}_{7,2,9,6,3,8,1,5,4,10,11}$ $+ y_5\mathbb{X}_{7,2,9,5,3,8,1,6,4,10,11}$
$\mathbb{X}_{7,2,9,6,3,8,1,4,5,10,11}$	$-\mathbb{X}_{7,6,9,2,3,8,1,4,5,10,11} + \mathbb{X}_{7,2,9,8,3,6,1,4,5,10,11} + y_6\mathbb{X}_{7,2,9,6,3,8,1,4,5,10,11}$
$\mathbb{X}_{7,2,8,9,3,6,1,4,5,10,11}$	$-\mathbb{X}_{7,2,9,8,3,6,1,4,5,10,11} + \mathbb{X}_{7,2,8,10,3,6,1,4,5,9,11} + y_9\mathbb{X}_{7,2,8,9,3,6,1,4,5,10,11}$

A noter que cette table contient des polynômes de Schubert affectés d'un signe négatif. Ceux-ci ne feront pas partie du résultat final d'après la proposition 5.7. En intégrant les résultats donnés par les deux tables précédentes, on obtient que $h_3(x_1, x_2, x_3, x_4)\mathbb{X}_\mu$ est égal à : $h_1(y_2, y_3, y_4, y_5, y_7, y_8)\mathbb{X}_{7,3,8,5,2,9,1,6,4} + h_1(y_2, y_3, y_4, y_6, y_7, y_8)\mathbb{X}_{7,3,8,6,2,9,1,4,5} + h_1(y_2, y_3, y_4, y_7, y_8, y_9)\mathbb{X}_{7,3,8,9,2,4,1,6,5} + h_1(y_2, y_4, y_6, y_7, y_8, y_9)\mathbb{X}_{7,2,9,6,3,8,1,4,5} + h_2(y_2, y_3, y_4, y_7,$

$y_8)\mathbb{X}_{7,3,8,4,2,9,1,6,5}+h_1(y_2,y_4,y_7,y_8,y_9,y_{10})\mathbb{X}_{7,2,10,4,3,8,1,6,5,9}+\mathbb{X}_{7,2,9,6,3,8,1,5,4}+\mathbb{X}_{7,3,8,6,2,9,1,5,4}+ h_1(y_2,y_3,y_4,y_7,y_8,y_9)\mathbb{X}_{7,3,9,4,2,8,1,6,5} + h_2(y_2,y_4,y_6,y_7,y_8)\mathbb{X}_{7,2,8,6,3,9,1,4,5} + \mathbb{X}_{7,3,8,9,2,6,1,4,5} + h_3(y_2,y_4,y_7,y_8)\,\mathbb{X}_{7,2,8,4,3,9,1,6,5} + \mathbb{X}_{7,2,11,4,3,8,1,6,5,9,10} + h_2(y_2,y_4,y_7,y_8,y_9)\,\mathbb{X}_{7,2,8,9,3,4,1,6,5} + h_2(y_2,y_4,y_5,y_7,y_8)\mathbb{X}_{7,2,8,5,3,9,1,6,4} + \mathbb{X}_{7,2,10,6,3,8,1,4,5,9} + \mathbb{X}_{7,2,10,5,3,8,1,6,4,9} + \mathbb{X}_{7,3,8,10,2,4,1,6,5,9} + \mathbb{X}_{7,3,8,9,2,5,1,6,4} + \mathbb{X}_{7,3,10,4,2,8,1,6,5,9} + h_1(y_2,y_4,y_6,y_7,y_8,y_9)\mathbb{X}_{7,2,8,9,3,6,1,4,5} + \mathbb{X}_{7,2,8,9,3,6,1,5,4} + \mathbb{X}_{7,2,8,10,3,6,1,4,5,9} + \mathbb{X}_{7,3,9,6,2,8,1,4,5} + \mathbb{X}_{7,2,8,11,3,4,1,6,5,9,10} + \mathbb{X}_{7,3,9,5,2,8,1,6,4} + \mathbb{X}_{7,2,8,10,3,5,1,6,4,9} + h_2(y_2,y_4,y_7,y_8,y_9)\,\mathbb{X}_{7,2,9,4,3,8,1,6,5} + h_1(y_2,y_4,y_7,y_8,y_9,y_{10})\,\mathbb{X}_{7,2,8,10,3,4,1,6,5,9} + h_1(y_2,y_4,y_5, y_7,y_8,y_9)\,\mathbb{X}_{7,2,8,9,3,5,1,6,4} + h_1(y_2,y_4,y_5,y_7,y_8,y_9)\,\mathbb{X}_{7,2,9,5,3,8,1,6,4} + h_1(y_2,y_4,y_5,y_6,y_7,y_8)\,\mathbb{X}_{7,2,8,6,3,9,1,5,4}.$

La propriété essentielle de la formule de Pieri double est comme dans le cas simple, l'absence de multiplicité. Par symétrie entre fonctions symétriques complètes et fonctions symétriques élémentaires, on peut donner le produit $\mathbf{Y}_{0^{p-i}1^i}\,\mathbb{X}_\mu$ qui fait intervenir des fonctions symétriques élémentaires comme coefficients des polynômes de Schubert dont les indices sont cette fois obtenus par produits de soulèvements gauches. Enfin, la proposition 5.7 nous précise que les permutations qui apparaissent dans (5.10) sont exactement celles impliquées dans les formules de Pieri pour les polynômes de Schubert simples :

$$\mathbf{Y}_{0^{p-1}m-j}\,\mathbf{X}_\mu \quad , \qquad j=0,\ldots,m \quad .$$

On peut noter que les seuls coefficients qui ne sont pas annulés par la spécialisation $y_i \longrightarrow 0$ sont les fonctions symétriques complètes $\mathbf{Y}_0 = 1$. En d'autres termes, la formule (5.10) nous précise qu'il faut interpréter les coefficients 1 de la formule (5.9) comme des fonctions symétriques complètes (resp. élémentaires) de degré 0.

5.4 Structure de module sur l'anneau des polynômes symétriques

Les calculs dans l'anneau $\mathbb{Z}[x_1,\ldots,x_n]$ des polynômes à plusieurs variables ne sont pas si aisés tant la taille des objets manipulés croît rapidement avec le nombre de variables et le degré. Un problème est par exemple de trouver des bases linéaires mieux adaptées à certains problèmes que la base usuelle des monômes, sans perdre pour autant la trace de la structure multiplicative. Dans le cas d'une variable, de telles bases sont fournies par la théorie des polynômes orthogonaux et la théorie de l'interpolation avec notamment l'interpolation de Lagrange ou l'interpolation de Newton.

Dans le cas de plusieurs variables, les outils sont beaucoup moins nombreux. Pourtant, en ce qui concerne les fonctions symétriques, les calculs sont usuellement faits en ayant recours à des objets combinatoires tels que les partitions ou les tableaux de Young, sans utiliser le développement de ces fonctions dans la base des monômes. Cela a de plus l'avantage de donner des identités indépendantes du nombre de variables. Malheureusement, ces méthodes ne peuvent être utilisées dans le cas de polynômes génériques, même si ceux-ci ont des symétries partielles.

En fait, l'action du groupe symétrique sur l'anneau des polynômes nous guide dans la recherche de nouvelles bases linéaires et les nombreuses propriétés combinatoires de ce groupe peuvent être exploitées au niveau des polynômes. Par exemple, Young au début du siècle a montré comment décomposer des polynômes en parties ayant un certain type de symétrie, décrivant en fait par là des idempotents de l'algèbre du groupe symétrique.

Pour être capable d'utiliser les méthodes liées aux fonctions symétriques dans le cas de polynômes génériques, nous regardons désormais l'anneau des polynômes en n variables comme module sur l'anneau $Sym[x_1, \ldots, x_n]$ des polynômes symétriques. Il est facile de montrer que l'ensemble des monômes de degré inférieur ou égal à $(n-1, \ldots, 2, 1, 0)$ composante à composante forme une base linéaire de ce module, qui est par ailleurs libre et de rang fini $n!$ [16, Chap. V]. Nous utiliserons cependant une base plus intéressante pour ce module libre sur l'anneau $Sym[\mathbf{x}] = Sym[x_1, \ldots, x_n]$, base qui consiste en les $n!$ polynômes de Schubert simples de $\mathfrak{S}_n$.

Proposition 5.9 *L'espace $\mathbb{Z}[\mathbf{x}]$ est un module libre de rang $n!$ sur l'anneau $Sym[\mathbf{x}]$ avec comme base les polynômes de Schubert simples $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$:*

$$\mathbb{Z}[x_1, \ldots, x_n] \quad = \quad \bigoplus_{\mu \in \mathfrak{S}_n} Sym[x_1, \ldots, x_n] \ \mathbf{X}_\mu \quad . \tag{5.11}$$

En d'autres termes, n'importe quel polynôme $p \in \mathbb{Z}[\mathbf{x}]$ peut s'exprimer de manière unique comme une somme :

$$p = \sum_{\mu \in \mathfrak{S}_n} c_\mu \, \mathbf{X}_\mu \quad ,$$

où les c_μ appartiennent à $Sym[\mathbf{x}]$.

Exemple 5.10 On considère un polynôme p de $\mathbb{Z}[x_1, \ldots, x_6]$ symétrique en $x_1, \ldots, x_5$. Comme nous l'avons déjà dit, le développement de p sur la base des monômes ne peut exploiter utilement cette symétrie des cinq premières variables. Pour illustrer notre propos, nous prenons l'exemple du polynôme : $p := x_6x_5^2x_4^2x_3^2x_2^2x_1 + x_6x_5^2x_4^2x_3^2x_2x_1^2 + x_6x_5^2x_4^2x_3x_2^2x_1^2 +$
$x_6x_5^2x_4x_3^2x_2^2x_1^2+x_6x_5x_4^2x_3^2x_2^2x_1^2+x_5^3x_4^3x_3x_2^3x_1+x_5^3x_4^3x_3x_2x_1^3+x_5^3x_4x_3^3x_2^3x_1+x_5^3x_4^2x_3^2x_2^3x_1+x_5^3x_4x_3^3x_2^2x_1^2+x_5^3x_4^3x_3^2x_2^2x_1+$
$x_5^3x_4^3x_3^2x_2x_1^2 + x_5^3x_4^3x_3x_2^2x_1^2 + x_5^3x_4^2x_3^3x_2^2x_1 + x_5^3x_4^2x_3^3x_2x_1^2 + x_5^3x_4^2x_3^2x_2x_1^3 + x_5^3x_4^2x_3x_2^3x_1^2 + x_5^3x_4x_3^2x_2^3x_1^2 + x_5^3x_4x_3^2x_2^2x_1^3 +$
$x_5^2x_4x_3^2x_2^3x_1^3 + x_5^2x_4^3x_3^3x_2^2x_1 + x_5^2x_4^3x_3^3x_2x_1^2 + x_5^2x_4^3x_3^2x_2^3x_1 + x_5^2x_4^3x_3^2x_2x_1^3 + x_5^2x_4^3x_3x_2^3x_1^2 + x_5^2x_4^3x_3x_2^2x_1^3 + x_5^2x_4^2x_3^3x_2^3x_1 +$
$x_5^2x_4^2x_3^3x_2x_1^3+x_5^2x_4^2x_3x_2^3x_1^3+x_5^2x_4x_3^3x_2^3x_1^2+x_5^2x_4x_3^3x_2^2x_1^3+2x_5^2x_4^3x_3^2x_2^2x_1^2+2x_5^2x_4^2x_3^3x_2^2x_1^2+2x_5^2x_4^2x_3^2x_2^3x_1^2+2x_5^2x_4^2x_3^2x_2^2x_1^3+$
$x_5x_4^3x_3^3x_2^2x_1^2 + x_5x_4^3x_3^2x_2^3x_1^2 + +x_5^3x_4x_3x_2x_1^3x_6^2 + x_5x_4x_3x_2^3x_1^3x_6^2 + x_5^3x_4^2x_3x_2^2x_1^3 + x_5^3x_4^2x_3x_2^2x_1x_6^2 + x_5^3x_4^2x_3x_2x_1^2x_6^2 +$
$x_5^3x_4x_3^2x_2x_1^2x_6^2+x_5^3x_4x_3x_2^2x_1^2x_6^2+x_5^2x_4x_3x_2^3x_1^2x_6^2+x_5^2x_4^3x_3x_2x_1^2x_6^2+x_5^2x_4^2x_3^3x_2x_1x_6^2+x_5^2x_4^2x_3x_2^3x_1x_6^2+x_5^2x_4^2x_3x_2x_1^3x_6^2+$
$x_5^2x_4x_3^3x_2^2x_1x_6^2+x_5^2x_4x_3^3x_2x_1^2x_6^2+x_5^2x_4x_3^2x_2^3x_1x_6^2+x_5x_4^3x_3^2x_2^2x_1x_6^2+x_5x_4^3x_3^2x_2x_1^2x_6^2+x_5x_4^3x_3x_2^2x_1^2x_6^2+x_5x_4^2x_3^3x_2^2x_1x_6^2+$
$x_5x_4^2x_3^3x_2x_1^2x_6^2+x_5x_4^2x_3^2x_2x_1^3x_6^2+x_5x_4^2x_3x_2^3x_1^2x_6^2+x_5x_4x_3^2x_2^3x_1^2x_6^2+x_5x_4x_3^2x_2^2x_1^3x_6^2+x_5^2x_4^3x_3x_2x_1^3x_6+x_5^2x_4x_3^3x_2^3x_1x_6+$
$x_5^2x_4^3x_3x_2^3x_1x_6+x_5^2x_4^3x_3^3x_2x_1x_6+x_5x_4x_3^2x_2^3x_1^3x_6+x_5^3x_4x_3^2x_2x_1^3x_6+x_5^3x_4^3x_3^2x_2x_1x_6+x_5^3x_4^3x_3x_2^2x_1x_6+x_5^3x_4x_3x_2^2x_1^3x_6+$
$x_5^3x_4x_3x_2^3x_1^2x_6+x_5^3x_4^3x_3x_2x_1^2x_6+x_5^3x_4^2x_3^3x_2x_1x_6+x_5^3x_4^2x_3x_2^3x_1x_6+x_5^3x_4^2x_3x_2x_1^3x_6+x_5^3x_4x_3^3x_2^2x_1x_6+x_5^3x_4x_3^3x_2x_1^2x_6+$
$x_5^3x_4x_3^2x_2^3x_1x_6+x_5^2x_4x_3^3x_2x_1^3x_6+x_5^2x_4x_3x_2^3x_1^3x_6+x_5x_4^3x_3^3x_2^2x_1x_6+x_5x_4^3x_3^3x_2x_1^2x_6+x_5x_4^3x_3^2x_2^3x_1x_6+x_5x_4^3x_3^2x_2x_1^3x_6+$
$x_5x_4^3x_3x_2^3x_1^2x_6+x_5x_4^3x_3x_2^2x_1^3x_6+x_5x_4^2x_3^3x_2^3x_1x_6+x_5x_4^2x_3^3x_2x_1^3x_6+x_5x_4^2x_3x_2^3x_1^3x_6+x_5x_4x_3^3x_2^3x_1^2x_6+x_5x_4x_3^3x_2^2x_1^3x_6+$
$x_5^3x_4x_3^2x_2^2x_1x_6^2+x_5^3x_4x_3^3x_2x_1x_6^2+x_5^3x_4x_3x_2^3x_1x_6^2+x_5x_4^2x_3x_2^2x_1^3x_6^2+x_5x_4^3x_3^3x_2x_1x_6^2+x_5x_4^3x_3x_2^3x_1x_6^2+x_5x_4^3x_3x_2x_1^3x_6^2+$
$x_5x_4x_3^3x_2x_1^3x_6^2+x_5^3x_4^3x_3^3x_2x_1+x_5^3x_4x_3^3x_2x_1^3+x_5^3x_4x_3x_2^3x_1^3+x_5x_4^3x_3^3x_2^3x_1+x_5x_4^3x_3^3x_2x_1^3+x_5x_4^3x_3x_2^3x_1^3+x_5^2x_4x_3^2x_2x_1^3x_6^2+$
$2x_5^2x_4^2x_3x_2^2x_1^2x_6^2 + 2x_5^2x_4x_3^2x_2^2x_1^2x_6^2 + x_5x_4x_3^3x_2^2x_1^2x_6^2 + x_5^3x_4^2x_3^2x_2x_1x_6^2 + x_5x_4^3x_3^2x_2^2x_1^3 + x_5x_4^2x_3^3x_2^3x_1^2 + x_5x_4^2x_3^3x_2^2x_1^3 +$
$x_5x_4x_3^3x_2^3x_1^3 + 5x_5^2x_4^2x_3^2x_2^2x_1^2x_6 + x_5x_4^2x_3^2x_2^3x_1^3 + x_5x_4x_3^3x_2^3x_1x_6^2 + x_5^3x_4^3x_3x_2x_1x_6^2 + 2x_5^2x_4^3x_3^2x_2^2x_1x_6 + 2x_5^2x_4^3x_3^2x_2x_1^2x_6 +$
$2x_5^2x_4^3x_3x_2^2x_1^2x_6+2x_5^2x_4^2x_3^3x_2^2x_1x_6+2x_5^2x_4^2x_3^3x_2x_1^2x_6+2x_5^2x_4^2x_3^2x_2x_1^3x_6+2x_5^2x_4^2x_3x_2^3x_1^2x_6+2x_5^2x_4x_3^2x_2^3x_1^2x_6+2x_5^2x_4x_3^2x_2^2$
$x_1^3x_6 + 2x_5x_4^3x_3^2x_2^2x_1^2x_6 + 2x_5x_4^2x_3^3x_2^2x_1^2x_6 + 2x_5x_4^2x_3^2x_2^3x_1^2x_6 + 2x_5x_4^2x_3^2x_2^2x_1^3x_6 + 2x_5^3x_4^2x_3^2x_2^2x_1^2 + 2x_5^3x_4^2x_3^2x_2^2x_1x_6 +$
$2x_5^3x_4^2x_3^2x_2x_1^2x_6+2x_5^3x_4^2x_3x_2^2x_1^2x_6+2x_5^3x_4x_3^2x_2^2x_1^2x_6+2x_5^2x_4^2x_3x_2^2x_1^3x_6+2x_5^2x_4^2x_3^2x_2^3x_1x_6+2x_5^2x_4x_3^3x_2^2x_1^2x_6+x_5^2x_4^3x_3^2x_2$
$x_1x_6^2 + x_5^2x_4^3x_3x_2^2x_1x_6^2 + x_5^2x_4x_3x_2^2x_1^3x_6^2 + x_5x_4^2x_3^2x_2^3x_1x_6^2 + 2x_5x_4^2x_3^2x_2^2x_1^2x_6^2 + 2x_5^2x_4^2x_3^2x_2^2x_1x_6^2 + 2x_5^2x_4^2x_3^2x_2x_1^2x_6^2.$

Ce polynôme a une expression plus compacte sur la base des polynômes de Schubert simples de $\mathfrak{S}_6$ avec comme coefficients des fonctions symétriques en $x_1, \ldots, x_6$ exprimées sur la base des fonctions de Schur :

$$p = s_{1^6}\mathbf{X}_{1,3,4,5,6,2} + s_{2^3}\mathbf{X}_{2,3,4,5,6,1} \quad .$$

On peut par exemple, si l'on sait décrire la structure multiplicative dans la base linéaire des $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$ de $\mathbb{Z}[\mathbf{x}]$, calculer le carré de ce polynôme : $p^2 = (2s_{4,4,4,2,1,1} + 2s_{4,4,3,2,2,1} + 2s_{4,3,3,2,2,2} + s_{3,3,3,2,2,2} + 2s_{3,3,3,3,3,2} + 2s_{4,3,3,3,2,2} + 2s_{4,4,3,2,2,2} + 2s_{4,3,3,3,3,1} + s_{5,3,3,3,3} + 2s_{5,3,3,3,2,1} + 2s_{4,4,3,3,2,1} + s_{5,4,3,3,2} + 2s_{5,4,3,2,2,1} + 2s_{4,4,4,2,2,1} + s_{5,4,4,2,2} + 2s_{5,4,3,3,1,1} + s_{5,5,3,3,1} + 2s_{5,5,3,2,1,1} + 2s_{5,4,4,2,1,1} + s_{5,5,4,2,1} + s_{5,5,5,1,1} + 2s_{5,5,4,1,1,1})\mathbf{X}_{2,3,4,5,6,1} + (s_{3,3,3,3,2,2} - s_{5,5,5,1,1,1} - s_{5,5,4,2,1,1} - s_{5,5,3,3,1,1} - s_{5,4,4,2,2,1} - s_{5,4,3,3,2,1} - s_{5,3,3,3,3,1} - s_{4,4,4,2,2,2} - s_{4,4,3,3,2,2} - s_{4,3,3,3,3,2} - s_{3,3,3,3,3,3})\mathbf{X}_{1,3,4,5,6,2} + (-2s_{4,4,4,2,2,2} - s_{3,3,3,3,3,2})\mathbf{X}_{1,2,4,5,6,3} - s_{3,3,3,3,3,3}\mathbf{X}_{1,2,3,5,6,4}$. Cette expression qui prend en compte la symétrie partielle de p nous évite de manipuler les 2117 monômes de p^2.

La formule de Monk (5.1) donne la structure multiplicative de l'anneau des polynômes en une infinité de variables mais elle ne préserve pas le sous-espace ayant comme base les $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$. En outre, elle n'utilise pas le fait que les coefficients peuvent appartenir à $Sym[\mathbf{x}]$. Par ailleurs, puisqu'il y a au plus un seul terme (dans la formule de Monk) indicé par une permutation ν en dehors de $\mathfrak{S}_n$, nous sommes naturellement conduit à exprimer certains polynômes de Schubert simples comme sommes à coefficients dans $Sym[\mathbf{x}]$ de polynômes de Schubert indicés par des permutations de $\mathfrak{S}_n$. Plus généralement, tout polynôme de Schubert de $\mathbb{Z}[\mathbf{x}]$ a une expression dans la base Schubert $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$.

Au lieu de donner des identités sur les polynômes de Schubert, nous allons donner des identités dans l'algèbre des différences divisées [13], aussi appelée algèbre nilCoxeter [3], qui vont impliquer les identités dont nous avons besoin. Nous rappelons que les polynômes symétriques en $x_1, \ldots, x_n$ sont des scalaires pour les opérateurs de différences divisées $\partial_1, \ldots, \partial_{n-1}$. Soient k, m deux entiers et A l'alphabet ordonné $A := \{\partial_1, \ldots, \partial_m\}$. On considère les fonctions symétriques non commutatives de A au sens de [6]. Grâce à des résultats de Lascoux et Schützenberger [14] et de Fomin et Greene [2], ces fonctions commutent en fait entre elles. On note $\Lambda^k(A)$ la k-ième fonction symétrique élémentaire :

$$\Lambda^k(A) := \sum_{i_1 > \cdots > i_k} \partial_{i_1} \ldots \partial_{i_k} \quad ,$$

et par $S^k(A)$ la k-ième fonction symétrique complète :

$$S^k(A) := \sum_{i_1 \leq \cdots \leq i_k} \partial_{i_1} \ldots \partial_{i_k} \quad = \quad \sum_{i_1 < \cdots < i_k} \partial_{i_1} \ldots \partial_{i_k} \quad ,$$

puisque $\partial_i^2 = 0$. On considère aussi les fonctions génératrices suivantes :

$$\lambda_t(A) = \sum \Lambda^i(A)\, t^i \quad , \tag{5.12}$$

et :

$$\sigma_t(A) = \sum S^i(A)\, t^i \quad . \tag{5.13}$$

Proposition 5.11 ([10], 5.1) *Soient m, n et k des entiers et soient $A := \{\partial_1, \ldots, \partial_m\}$ et $B := \{\partial_{m+1}, \ldots, \partial_{m+n}\}$. Nous notons $A + B := \{\partial_1, \ldots, \partial_m, \partial_{m+1}, \ldots, \partial_{m+n}\}$. Alors, dans l'algèbre nilCoxeter :*

$$\Lambda^k(B) = \Lambda^k(A+B)S^0(A) - \Lambda^{k-1}(A+B)S^1(A)\Lambda^{k-2}(A+B)S^2(A) - \cdots \quad . \tag{5.14}$$

Preuve. On a $\sigma_t(A+B) \stackrel{\text{def}}{=} \sigma_t(A)\,\sigma_t(B)$, de sorte que $\lambda_t(A+B) = \lambda_t(B)\,\lambda_t(A)$. Ainsi :

$$\lambda_t(B) = \lambda_t(A+B)\,\lambda_t^{-1}(A) = \lambda_t(A+B)\,\sigma_{-t}(A) \quad .$$

En extrayant le coefficient de t^k des deux côtés, on obtient (5.14). ■

Etant donné un mot $w := \partial_i\partial_j\ldots\partial_k$, on note w^+ le mot $\partial_{i+1}\partial_{j+1}\ldots\partial_{k+1}$ et on étend ce morphisme à l'algèbre nilCoxeter par linéarité.

Proposition 5.12 ([10], 5.2) *Soient m,k deux entiers et soient $A := \{\partial_1,\ldots,\partial_m\}$ et $B := \{\partial_{m+1},\ldots,\partial_{m+k}\}$. Alors, dans l'algèbre nilCoxeter :*

1. *il existe des éléments $T_{i,m}$ tels que :*

$$\partial_{m+k}\ldots\partial_{m+1} = \sum_{i=k}^{0}(-1)^{k-i}\,\Lambda^i(A+B)\,T_{i,m} \quad , \tag{5.15}$$

qui satisfont la récurrence suivante :

$$T_{i,m+1} = T_{i,m} - T_{i+1,m}\,\partial_{m+1} \quad ; \tag{5.16}$$

2. *il existe des éléments $U_{i,m}$ satisfaisant les relations suivantes :*

$$\partial_1\ldots\partial_k = \sum_{i=k}^{0}(-1)^{k-i}\,S^i(A+B)\,U_{i,m} \quad , \tag{5.17}$$

$$U_{i,m+1} = U_{i,m} - \partial_{m+1+k}\,U_{i+1,m} \quad ; \tag{5.18}$$

3. *il existe des éléments $V_{i,m}$ satisfaisant les relations suivantes :*

$$\partial_{m+1}\ldots\partial_{m+k} = \sum_{i=k}^{0}(-1)^{k-i}\,S^i(A+B)\,V_{i,m} \quad , \tag{5.19}$$

$$V_{i,m+1} = V_{i,m}^{+} - V_{i+1,m}^{+}\,\partial_1 \quad ; \tag{5.20}$$

4. *finalement, il existe des éléments $W_{i,m}$ tels que :*

$$\partial_1\ldots\partial_k = \sum_{i=k}^{0}(-1)^{k-i}\,W_{i,m}\,S^i(A+B) \quad , \tag{5.21}$$

et qui satisfont la récurrence (5.18).

Preuve. (5.15) est un cas particulier de la proposition 5.11 pour $k = n$. (5.17) lui est équivalent modulo le renversement de l'ordre des différences divisées. (5.16) et (5.18) sont équivalentes pour la même raison. Dit autrement, on a deux récurrences et nous allons seulement montrer la formule pour $\partial_1\ldots\partial_k$ car dans ce cas, la preuve consiste à contrôler que les termes supplémentaires, lorsque l'on passe de m à $m+1$, s'annulent les uns avec les autres de telle sorte que l'égalité soit préservée.

Ceci est une conséquence de la récurrence sur les ensembles de mots comme nous allons le vérifier pour des raisons de simplicité pour (5.21) dans le cas $k = 2$. Les récurrences sur les W_i étant simplement des récurrences sur les mots, nous omettons d'écrire le signe ∂ : à partir de maintenant, 12 dénote $\partial_1\partial_2$. Supposons par hypothèse de récurrence que la formule est vérifiée pour $k+m=4$, montrons la pour $k+m=5$. Si l'on vérifie que :

$$12 = S^2(1,2,3,4) - (3+4)S^1(1,2,3,4) + 43 + 32 - 23 \quad ,$$

alors la proposition nous dit qu'en appliquant la règle $W_{i,m+1} = W_{i,m} - (m+1+k)\, W_{i+1,m}$, on a encore l'égalité suivante :

$$12 = S^2(1,2,3,4,\mathbf{5}) - (3+4+\mathbf{5})S^1(1,2,3,4,\mathbf{5}) + \mathbf{5}(3+4) + 43 + 32 - 23 \quad .$$

Maintenant, l'expression impliquant la lettre **5** est égale à $S^1(1,2,3,4)\mathbf{5} - \mathbf{5}S^1(1,2,3,4,\mathbf{5}) - (3+4)\mathbf{5} + \mathbf{5}3 + \mathbf{5}4$, dont le développement est $1\mathbf{5} + 2\mathbf{5} + 3\mathbf{5} + 4\mathbf{5} - \mathbf{5}1 - \mathbf{5}2 - \mathbf{5}3 - \mathbf{5}4 - \mathbf{55} - 3\mathbf{5} - 4\mathbf{5} + \mathbf{5}3 + \mathbf{5}4 = 0$. ∎

Chaque identité sur les mots de la proposition 5.12 peut être traduite en termes de polynômes de Schubert simples. Le problème que nous considérons est d'exprimer les polynômes de Schubert $\mathbf{X}_{\mu'}$ indicés par des permutations $\mu' \in \mathfrak{S}_{n+1} \backslash \mathfrak{S}_n$ en termes de $\mathbf{X}_\mu, \mu \in \mathfrak{S}_n$. Les propriétés des polynômes de Schubert nous conduisent à ne considérer que les permutations μ' de type $(n+1, n, \ldots, \widehat{m}, \ldots, 2, 1, m)$ car en utilisant les différences divisées $\partial_1, \ldots, \partial_{n-1}$, on a tous les autres polynômes de Schubert. Pour ces permutations spéciales μ', on a l'identité suivante :

$$\partial_{m+k} \ldots \partial_{m+1} \left(\mathbf{X}_{m+k+1,\ldots,2,1}\right) = \mathbf{X}_{m+k+1,\ldots,\widehat{k+1},\ldots,2,1,k+1} \quad ,$$

qui implique la partie gauche de (5.20).

Finalement, on a aussi besoin du lemme suivant qui relie la multiplication des polynômes de Schubert simples par les $e_i(x_1, \ldots, x_n)$ à la multiplication par les $\Lambda^i(\partial_1, \ldots, \partial_n)$ dans l'algèbre nilCoxeter.

Lemme 5.13 ([10], 5.3) *Soient $\mu \in \mathfrak{S}_n$ et w_n (resp. w_{n+1}) l'élément maximal de $\mathfrak{S}_n$ (resp. $\mathfrak{S}_{n+1}$). Alors :*

$$e_i(x_1, \ldots, x_n)\, \mathbf{X}_\mu = \partial_{\mu^{-1}w_n}\, \Lambda^{n-i}(\partial_1, \ldots, \partial_n)\, \mathbf{X}_{w_{n+1}} \quad . \tag{5.22}$$

Preuve. Puisque $\mathbf{X}_\mu = \partial_{\mu^{-1}w_n}\mathbf{X}_{w_n}$ et puisque $\partial_{\mu^{-1}w_n}$ commute avec la multiplication par $e_i(x_1, \ldots, x_n)$, il est suffisant de prouver la formule pour $\mu = w_n$. Nous allons le faire pour tous les $e_i(x_1, \ldots, x_n)$ en même temps. La somme $\sum e_i(x_1, \ldots, x_n)\, \mathbf{X}_{w_n}$ est égale à la somme de tous les monômes dont les exposants sont entre $(n-1, n-2, \ldots, 1, 0, 0)$ et $(n, n-1, \ldots, 1, 0)$. Il en va de même pour la somme $\sum \Lambda^i(\partial_1, \ldots, \partial_n)\, \mathbf{X}_{w_{n+1}}$ car l'ordre selon lequel s'appliquent les différences divisées est tel que, à chaque étape on effectue le calcul $\partial_j\left(\ldots x_j^{k+1} x_{j+1}^k \ldots\right) = \ldots x_j^k x_{j+1}^k \ldots$, qui est un monôme. ∎

Nous sommes désormais en mesure de donner un algorithme directement sur les permutations. Pour cela, on considère les opérateurs de $\mathfrak{S}_{n-1}$ dans $\mathfrak{S}_n$, $P_0^n, \ldots, P_k^n$ définis comme suit :

$$\begin{aligned} P_0^n(\mu) &\longrightarrow [n, \mu_1, \mu_2, \ldots] \\ P_1^n(\mu) &\longrightarrow [\mu_1, n, \mu_2, \ldots] \\ \vdots &\longrightarrow \quad \vdots \\ P_k^n(\mu) &\longrightarrow [\mu_1, \ldots, \mu_{k-1}, n, \mu_k, \ldots] \end{aligned}$$

Proposition 5.14 ([10], 5.4) *Soient n, m deux entiers tels que $m \leq n$. Soit $\mathcal{R}(n,m)$ le polynôme défini récursivement par :*

$$\mathcal{R}(n,m) := \begin{cases} e_1 \mathbf{X}_{n,n-1,\ldots,2,1} - \sum_{k=1}^{n-1} P_{n-k}^n \Big(\mathcal{R}(n-1,k) \Big)_{e_j \leftarrow e_{j+1}} \quad , & \text{si } m = n, \\ \\ P_0^n \Big(\mathcal{R}(n-1,m) \Big)_{e_j \leftarrow e_{j+1}} - P_1^n \Big(\mathcal{R}(n-1,m) \Big)_{e_j \leftarrow e_{j+2}} \quad , & \text{autrement.} \end{cases}$$

Alors, $\mathcal{R}(n,m)$ est égal au polynôme de Schubert simple indicé par la permutation $[n+1, n, \ldots, \widehat{m}, \ldots, 2, 1, m]$.

Preuve. La preuve découle des propositions précédentes. ■

Une propriété intéressante de cette proposition est que les seules fonctions symétriques apparaissant dans ces développements sont les fonctions élémentaires. Utilisant les différences divisées $\partial_1, \ldots, \partial_{n-1}$ qui commutent avec les e_i, on déduit de la proposition 5.14 une expression de tout $\mathbf{X}_\mu, \mu \in \mathfrak{S}_{n+1} \backslash \mathfrak{S}_n$.

Exemple 5.15 Nous donnons quelques exemples d'utilisation de la proposition 5.14 en montrant explicitement la récurrence sur les polynômes $\mathcal{R}(n,m)$. Nous venons de voir que le polynôme $\mathcal{R}(n,m)$ est égal au polynôme de Schubert indicé par la permutation :

$$[\mathbf{n}+1, n, \ldots, \widehat{m}, \ldots, 2, 1, \mathbf{m}] \quad .$$

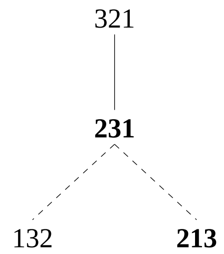
Par exemple, pour $m = \mathbf{3}$ et $n = 3, 4$, on obtient :

$$\begin{aligned} \mathbf{X}_{4,2,1,\mathbf{3}} &= (\mathbf{X}_{1,3,2} - \mathbf{X}_{2,1,3}) e_3(x_1, x_2, x_3) \\ &\quad + (-\mathbf{X}_{2,3,1}) e_2(x_1, x_2, x_3) \\ &\quad + (\mathbf{X}_{3,2,1}) e_1(x_1, x_2, x_3) \\ \\ \mathbf{X}_{5,4,2,1,\mathbf{3}} &= (\mathbf{X}_{\mathbf{4},1,3,2} - \mathbf{X}_{\mathbf{4},2,1,3} + \mathbf{X}_{2,\mathbf{4},3,1}) e_4(x_1, x_2, x_3, x_4) \\ &\quad + (-\mathbf{X}_{\mathbf{4},2,3,1} - \mathbf{X}_{3,\mathbf{4},2,1}) e_3(x_1, x_2, x_3, x_4) \\ &\quad + (\mathbf{X}_{\mathbf{4},3,2,1}) e_2(x_1, x_2, x_3, x_4) \end{aligned}$$

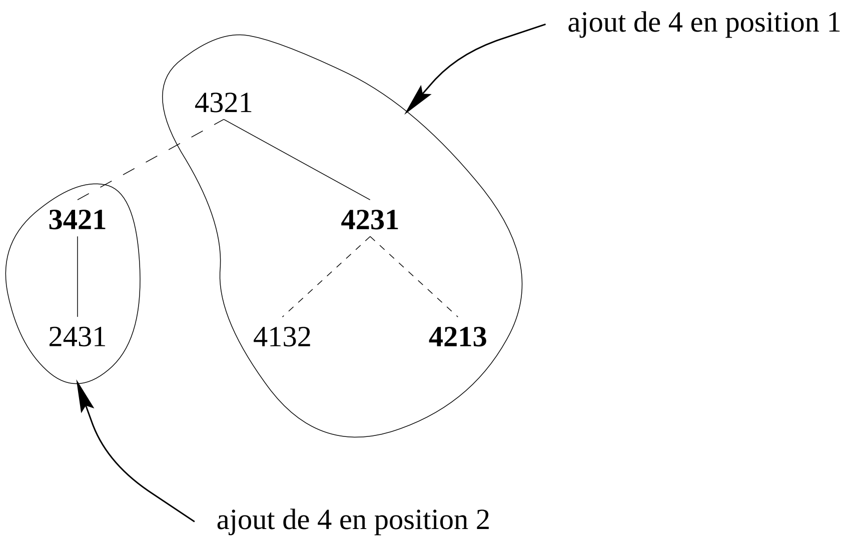
Le passage de l'expression de $\mathbf{X}_{4,2,1,\mathbf{3}}$ à $\mathbf{X}_{5,4,2,1,\mathbf{3}}$ est donné par la récurrence de la proposition 5.14 qui se voit plus facilement graphiquement. Nous donnons ci-après trois arbres représentant respectivement les expressions des polynômes de Schubert $\mathbf{X}_{4,2,1,\mathbf{3}}$, $\mathbf{X}_{5,4,2,1,\mathbf{3}}$ et $\mathbf{X}_{6,5,4,2,1,\mathbf{3}}$ afin de mettre en évidence la récurrence entre $\mathcal{R}(3,\mathbf{3})$, $\mathcal{R}(4,\mathbf{3})$ et $\mathcal{R}(5,\mathbf{3})$.

Les sommets des arbres suivants sont les permutations qui indicent les polynômes de Schubert dans ces trois expressions. Ils représentent donc des polynômes de Schubert organisés par niveau en fonction du degré de la fonction élémentaire qu'ils ont en coefficient. Nous n'avons pas fait figurer ces coefficients puisqu'il est simple de les retrouver pour des raisons d'homogénéité. Enfin, les permutations en gras sont affectées d'un signe négatif :

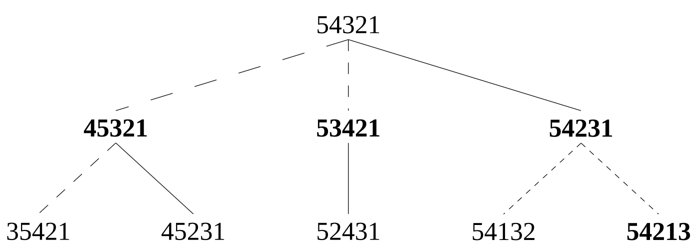
$\mathbf{X}_{4,2,1,\mathbf{3}} =$



$\mathbf{X}_{5,4,2,1,\mathbf{3}} =$



$\mathbf{X}_{6,5,4,2,1,\mathbf{3}} =$



Ces trois arbres illustrent la récurrence :

$$\mathcal{R}(n,m) = P_0^n\Big(\mathcal{R}(n-1,m)\Big)_{e_j\leftarrow e_{j+1}} - P_1^n\Big(\mathcal{R}(n-1,m)\Big)_{e_j\leftarrow e_{j+2}} \quad , \qquad \text{si } m \neq n,$$

en détaillant l'action des opérateurs P_0^n (ajout de n en première position) et P_1^n (ajout de n en deuxième position). Le dernier arbre code la décomposition cherchée :

$$\begin{aligned}\mathbf{X}_{6,5,4,2,1,\mathbf{3}} &= (\mathbf{X}_{5,4,1,3,2} + \mathbf{X}_{4,5,2,3,1} - \mathbf{X}_{5,4,2,1,3} + \mathbf{X}_{5,2,4,3,1} + \mathbf{X}_{3,5,4,2,1})e_5(x_1,\ldots,x_5)\\ &\quad -(\mathbf{X}_{5,3,4,2,1} + \mathbf{X}_{5,4,2,3,1} + \mathbf{X}_{4,5,3,2,1})e_4(x_1,\ldots,x_5)\\ &\quad +\mathbf{X}_{5,4,3,2,1}e_3(x_1,\ldots,x_5)\end{aligned}$$

5.5 Formule de Pieri quantique

Les géomètres ont obtenu une q-déformation de l'anneau de cohomologie de la variété de drapeaux [7]. Combinatoirement, l'anneau de cohomologie quantique de la variété de drapeaux $\mathcal{F}(\mathbb{C}^n)$ se décrit comme un quotient de l'anneau des polynômes en $x_1,\ldots,x_n$ par une q-déformation de l'idéal engendré par les fonctions élémentaires en $x_1,\ldots,x_n$. Plus précisément, les fonctions symétriques élémentaires peuvent être vues comme les coefficients du polynôme caractéristique :

$$\det(M - \lambda\, Id) = \prod(\lambda - x_i) \quad ,$$

de la matrice :

$$M = \begin{bmatrix} x_1 & 0 & 0 & \cdots & 0\\ -1 & x_2 & 0 & \cdots & 0\\ 0 & -1 & x_3 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & x_n \end{bmatrix} \quad .$$

Les q-déformations des fonctions symétriques élémentaires sont les coefficients $E_0,\ldots,E_n$ du polynôme caractéristique :

$$\det(M_q + \lambda\, Id) = \sum_{i=0}^{n} \lambda^{n-k}\, E_k(x_1,\ldots,x_n) \quad ,$$

du q-analogue M_q de la matrice M :

$$M_q = \begin{bmatrix} x_1 & q_1 & 0 & \cdots & 0\\ -1 & x_2 & q_2 & \cdots & 0\\ 0 & -1 & x_3 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & x_n \end{bmatrix} \quad .$$

Les $E_k(x_1,\ldots,x_n)$ sont appelées *fonctions élémentaires quantiques*. L'anneau de cohomologie quantique est défini comme :

$$\mathbb{Z}[x_1,\ldots,x_n][q_1,\ldots,q_n]\big/\mathcal{I}_q \quad ,$$

où $\mathcal{I}_q$ dénote l'idéal engendré par les $E_k(x_1, \ldots, x_n)$. Une base linéaire de cet anneau qui correspond aux variétés de Schubert, a été donnée dans [1] et [8]. Il existe un produit scalaire sur l'anneau de cohomologie usuel, pour lequel la base des polynômes de Schubert est auto-adjointe et la base $\{P_{\mathbf{i}}\}$ est adjointe de la base des monômes $\mathbf{x}^{\mathbf{i}'} : x_2^{1-i_{n-1}} \ldots x_n^{n-1-i_1}$. A l'aide de ce produit scalaire, les polynômes de Schubert peuvent donc s'écrire :

$$\mathbf{X}_\mu = \sum_{\mathbf{i}} (\mathbf{X}_\mu, \mathbf{x}^{\mathbf{i}'}) \, P_{\mathbf{i}} \quad ,$$

avec $\mu \in \mathfrak{S}_n$ et $\mathbf{i} \in \mathbb{N}^n$. Notons à ce propos que la proposition 5.14 fournit un algorithme de décomposition d'un polynôme de Schubert sur la base des produits de fonctions élémentaires. En effet, en itérant la récurrence sur les polynômes $\mathcal{R}(n, m)$, on fait apparaître successivement dans le membre droit de l'équation :

$$\mathcal{R}(n,m) := \begin{cases} e_1 \mathbf{X}_{n,n-1,\ldots,2,1} - \displaystyle\sum_{k=1}^{n-1} P^n_{n-k}\Big(\mathcal{R}(n-1,k)\Big)_{e_j \leftarrow e_{j+1}} \quad , & \text{si } m = n, \\[2ex] P^n_0\Big(\mathcal{R}(n-1,m)\Big)_{e_j \leftarrow e_{j+1}} - P^n_1\Big(\mathcal{R}(n-1,m)\Big)_{e_j \leftarrow e_{j+2}} \quad , & \text{autrement.} \end{cases}$$

des fonctions symétriques élémentaires en n variables, puis en $n-1$ variables, et ainsi de suite. Par exemple, $\mathbf{X}_{3,1,5,4,2}$ est par la proposition 5.14 égal à $-e_4(x_1, x_2, x_3, x_4)\mathbf{X}_{1,2,4,3} + e_3(x_1, x_2, x_3, x_4)\mathbf{X}_{2,1,4,3} - e_4(x_1, x_2, x_3, x_4)\mathbf{X}_{2,1,3,4}$. Mais $\mathbf{X}_{1,2,4,3} = e_1(x_1, x_2, x_3)$, $\mathbf{X}_{2,1,4,3} = x_1 \mathbf{X}_{1,2,4,3} = e_1(x_1)e_1(x_1, x_2, x_3)$ et $\mathbf{X}_{2,1,3,4} = e_1(x_1)$.

Par suite, $\mathbf{X}_{3,1,5,4,2}$ est égal à $-e_4(x_1, x_2, x_3, x_4)e_1(x_1, x_2, x_3) + e_3(x_1, x_2, x_3, x_4)e_1(x_1, x_2, x_3)e_1(x_1) - e_4(x_1, x_2, x_3, x_4)e_1$ ce que l'on note :

$$\mathbf{X}_{3,1,5,4,2} = -P_{4,1,0,0,0,0} + P_{3,1,0,1,0} - P_{4,0,0,1,0} \quad .$$

Fomin, Gelfand et Postnikov [1] ont introduit des monômes standards quantiques $P^q_{\mathbf{i}}$ qui désignent les q-déformations de $P_{\mathbf{i}}$ obtenues en remplaçant les fonctions élémentaires par les fonctions élémentaires quantiques. Ils ont alors définis les *polynômes de Schubert quantiques* $\mathbf{X}^q_\mu$ par la formule :

$$\mathbf{X}^q_\mu = \sum_{\mathbf{i}} (\mathbf{X}_\mu, \mathbf{x}^{\mathbf{i}'}) P^q_{\mathbf{i}} \quad ,$$

qui forment une base de l'anneau quantique. Kirillov et Maeno [8] partent de la fonction génératrice :

$$\mathbb{X}^q_\omega := \sum \mathbf{y}^{\mathbf{i}'} P^q_{\mathbf{i}} = \sum \mathbf{X}_{\mu\omega}(\mathbf{y}) \mathbf{X}^q_\mu(\mathbf{x}) \quad ,$$

qui implique la formule précédente, mais montre aussi que les polynômes quantiques peuvent se calculer à l'aide de différences divisées agissant sur les variables $\mathbf{y}$. En fait, ils définissent les polynômes de Schubert doubles quantiques comme image de $\mathbb{X}^q_\omega$ par les différences divisées en $\mathbf{y}$, les polynômes de Fomin, Gelfand et Postnikov s'obtenant par la spécialisation $\mathbf{y} = 0$.

Exemple 5.16 Par exemple, le polynôme de Schubert quantique $\mathbf{X}^q_{3,1,5,4,2}$ est par définition égal à :

$$\mathbf{X}^q_{3,1,5,4,2} = -P^q_{4,1,0,0,0,0} + P^q_{3,1,0,1,0} - P^q_{4,0,0,1,0} \quad ,$$

c'est-à-dire : $x_2^2x_1^2x_3 + x_2^2x_1^2x_4 + x_2x_1^3x_3 + x_2x_1^3x_4 + x_2x_1^2x_3^2 + x_2x_1^2x_3x_4 + x_1^3x_3x_4 + x_1^2x_3^2x_4 + q_2x_2x_1^2 + q_1x_2x_1x_3 + q_1x_2x_1x_4 - q_1x_2x_3x_4 + q_2x_1^3 + q_3x_1^3 + q_1x_1^2x_3 + q_1x_1^2x_4 + q_2x_1^2x_3 - q_2x_1^2x_4 + q_3x_1^2x_3 + q_1x_1x_3^2 - q_1x_1x_3x_4 - q_1x_3^2x_4 - q_1q_3x_2 - 2q_1q_3x_1 - q_1q_3x_3$.

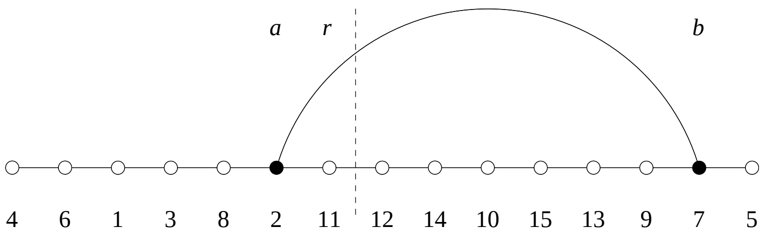
Ils ont donné la formule de Monk pour les polynômes de Schubert quantiques et ainsi caractérisé la structure multiplicative de l'anneau quantique.

Proposition 5.17 ([1], 5.3)

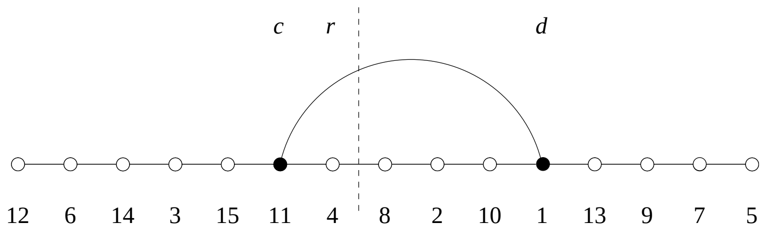
$$(x_1 + \cdots + x_r)\mathbf{X}_\mu^q = \sum \mathbf{X}_{\mu t_{ab}}^q + \sum q_{cd}\mathbf{X}_{\mu t_{cd}}^q \quad , \tag{5.23}$$

où la première somme est sur toutes les transpositions t_{ab}, $a \le r < b$, $l(\mu t_{ab}) = l(\mu) + 1$, la deuxième somme porte sur toutes les transpositions t_{cd}, $c \le r < d$, $l(\mu t_{cd}) = l(\mu) - l(t_{cd}) = l(\mu) - 2(d - c) + 1$ et où $q_{cd} := q_c q_{c+1} \cdots q_{d-1}$.

Cette formule met en jeu deux opérations élémentaires différentes qui correspondent respectivement à la première et à la deuxième somme. En effet, la première somme correspond à des transpositions augmentant la longueur de la permutation μ de 1. Ce sont les mêmes transpositions qui interviennent dans la formule de Monk classique et nous les appelons désormais *étapes de Monk*. La figure suivante en donne un exemple concret :



Soient $\mu_a < \mu_b$ les deux entrées modifiées par la transposition t_{ab} de la figure précédente. Alors, il est clair que $l(\mu t_{ab}) = l(\mu) + 1$ si et seulement pour tout i, $a < i < b$, on a soit $\mu_i > \mu_b$ soit $\mu_i < \mu_a$. La deuxième somme correspond à des transpositions t_{cd} qui diminuent la longueur de la permutation μ. Plus précisément, $\mu_c > \mu_d$ et pour tout i, $c < i < d$, on a $\mu_d < \mu_i < \mu_c$. On appelle *saut quantique* une telle transposition, la figure suivante correspondant à ce cas :



puisque $2, 4, 8$ et 10 sont physiquement entre les positions c et d et ces valeurs sont bien comprises entre $\mu_c = 11$ et $\mu_d = 1$.

Exemple 5.18 Par exemple, soit μ la permutation $[2, 5, 4, 3, 1]$, alors :

$$\begin{aligned}\mathbf{X}_{1,2,4,3}^q\mathbf{X}_{2,5,4,3,1}^q &= \mathbf{X}_{2,5,6,3,1,4}^q + \mathbf{X}_{2,6,4,3,1,5}^q + \mathbf{X}_{3,5,4,2,1}^q \\ &\quad + q_3\mathbf{X}_{2,5,3,4,1}^q + q_2q_3\mathbf{X}_{2,3,4,5,1}^q + q_3q_4\mathbf{X}_{2,5,1,3,4}^q + q_2q_3q_4\mathbf{X}_{2,1,4,3}^q \quad .\end{aligned}$$

Puisque la base est définie en termes de $P^q_{\mathbf{i}}$, il est utile de disposer d'une formule de Pieri quantique exprimant le produit d'un polynôme de Schubert quantique par une fonction élémentaire ou complète quantique. Dans un récent travail, nous avons obtenu avec Lascoux la formule de Pieri quantique [15]. Celle-ci s'exprime à nouveau en termes d'opérations élémentaires qui peuvent être vues comme des q-déformations des soulèvements impliqués dans la formule de Pieri pour les polynômes de Schubert.

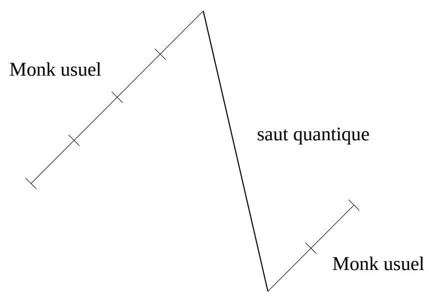
Comme précédemment, nous introduisons l'étape élémentaire impliquée dans la formule de Pieri pour les polynômes de Schubert quantiques et nous l'appelons (k,p)-soulèvement quantique. Soient p et k deux entiers positifs et $\mu \in \mathfrak{S}_n$. Alors, un (k,p)-soulèvement quantique droit est un cycle ζ tel que :

① μ et $\mu\zeta = \nu$ diffèrent en $k+1$ positions $i_0, \ldots, i_k$, telles que $i_0 \leq p$, $p+1 \leq i_1 \leq \cdots \leq i_k$;

② ζ est réalisé de manière unique comme une suite de $m_1 \geq 0$ étapes de Monk suivie d'au plus un saut quantique et d'une suite de $m_2 \geq 0$ étapes de Monk ;

③ soit t_{cd} le saut quantique potentiel. S'il est présent alors $l(\mu\zeta) = l(\mu) + m_1 + m_2 - 2(d-c) + 1$ et sinon $l(\mu\zeta) = l(\mu) + m_1 + m_2$.

Un (k,p)-soulèvement quantique d'une permutation μ revient à augmenter la longueur de μ puis à la diminuer au plus une fois et finalement à l'augmenter à nouveau comme l'illustre la figure suivante :



A chaque saut quantique t_{cd}, on associe le produit q_{cd} et plus généralement à chaque produit ζ de soulèvements quantiques ζ_i, on associe le produit des q_{cd} issu de chaque soulèvement quantique et on note ce produit global q_ζ. Notons que s'il n'y a aucun saut quantique dans ζ alors $q_\zeta = 1$. On pose degré de $q_i = 2$ pour tout i. Dans [15], nous avons établi avec Lascoux la proposition suivante :

Proposition 5.19 (formule de Pieri quantique)

$$\mathbf{Y}^q_{0^{p-1}m} \mathbf{X}^q_\mu = \sum_\nu q_\zeta \mathbf{X}^q_\nu \quad , \tag{5.24}$$

où la somme est étendue à toutes les permutations ν telles que la décomposition en cycles de $\nu\mu^{-1}$ ne comporte que des (i,p)-soulèvements quantiques droits ζ_i avec $l(\nu) + deg(q_\zeta) = l(\mu) + m$.

Preuve. La preuve se fait par récurrence au moyen de l'identité sur les fonctions complètes quantiques :

$$\mathbf{Y}^q_{0^{p-1}m} = \mathbf{Y}^q_{0^{p-2}m} + x_p \mathbf{Y}^q_{0^{p-1}m-1} - q_p \mathbf{Y}^q_{0^p m-2} \quad .$$

∎

Nous venons de recevoir une prépublication de Kirillov et Maeno [8] qui énoncent une règle de multiplication d'un polynôme de Schubert quantique par une fonction élémentaire quantique.

Références bibliographiques

[1] S. Fomin, S. Gelfand, and A. Postnikov. Quantum Schubert polynomials. (1996).

[2] S. Fomin and C. Greene. Noncommutative Schur functions and their applications, (1995). preprint.

[3] S. Fomin and A. Kirillov. Yang-Baxter equation, symmetric functions and Schubert polynomials. In *Proceedings of the 5-th Conference "Formal Power Series and Algebraic Combinatorics"*, pages 215–229, Florence, (1993).

[4] W. Fulton. Flags, Schubert polynomials, degeneracy loci and determinantal formulas. *Duke Math. J.*, 65: 381–420, (1991).

[5] W. Fulton and A. Lascoux. A Pieri formula in the Grothendieck ring of a flag bundle. *Duke Mathematical Journal*, 76: 711–729, (1994).

[6] I.M. Gelfand, D. Krob, A. Lascoux, B. Leclerc, V.S. Retakh, and J.-Y. Thibon. Noncommutative symmetric functions. *Advances in Mathematics*, 112: 218–348, (1995).

[7] A. Givental and B. Kim. Quantum cohomology of flag manifolds and Toda lattices. *Commun. Math. Phys.*, 168: 609–641, (1995).

[8] A.N. Kirillov and T. Maeno. Quantum Schubert polynomials and the Vafa-Intriligator formula. (1996).

[9] A. Kohnert and S. Veigneau. Using Schubert basis to compute with multivariate polynomials. *Advances in Applied Mathematics*, (1996). à paraître.

[10] A. Kohnert and S. Veigneau. Using Schubert toolkit to compute with polynomials in several variables. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 283–293, Minneapolis, (1996). University of Minnesota.

[11] A. Lascoux. Classes de Chern des variétés de drapeaux. *C.R. Acad. Sci. Paris*, 295: 393–398, (1982).

[12] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

[13] A. Lascoux and M.P. Schützenberger. Structure de Hopf de l'anneau de cohomologie et de l'anneau de Grothendieck d'une variété de drapeaux. *C.R. Acad. Sci. Paris*, 295: 629–633, (1982).

[14] A. Lascoux and M.P. Schützenberger. Fonctorialité des polynômes de Schubert. *Contempory Maths*, 88: 585–598, (1989).

[15] A. Lascoux and S. Veigneau. Double and quantum Pieri formula. (1996).

[16] I.G. Macdonald. *Notes on Schubert polynomials*, volume 6. Publ. LACIM, Montréal, (1991).

[17] D. Monk. The Geometry of flag manifolds. *Proc. London M.S. (3)*, 9: 253–286, (1959).

[18] F. Sottile. A Geometric Approach to the Combinatorics of Schubert Polynomials. (1994).

Conclusion

Voilà le commencement de la fin.

— ***Charles-Maurice de Talleyrand-Périgord***, *Attribué à Talleyrand.*

La combinatoire des fonctions symétriques et plus particulièrement des fonctions de Schur est très riche. En fait, ces dernières ont été introduites il y a 150 ans par Jacobi et Cauchy et constituent encore un domaine en pleine activité. Comme les polynômes de Schubert contiennent comme sous-famille les fonctions de Schur, on peut donc s'attendre à ce que leur combinatoire suscite des travaux pendant de nombreuses années.

Les polynômes de Schubert ont été définis au début des années 80 mais c'est à partir des notes de Macdonald qu'ils ont commencé à susciter un intérêt plus développé. En témoignent les généralisations que sont les polynômes de Schubert associés aux groupes hyperoctaédraux et les polynômes de Schubert quantiques.

Définis initialement pour résoudre un problème de géométrie algébrique, les polynômes de Schubert ont rapidement trouvé des applications dans d'autres domaines. Par exemple, nous avons vu qu'ils sont les coefficients universels dans la généralisation à plusieurs variables de la formule d'interpolation de Newton. Ils sont donc un outil fondamental en interpolation et en calcul formel (calcul polynomial à plusieurs variables). Ils interviennent aussi dans des domaines très nouveaux comme la cohomologie quantique.

Nos travaux ont spécialement privilégié les polynômes de Schubert comme base distinguée de l'anneau des polynômes en tant que module libre sur l'anneau des polynômes symétriques. Ceci permet d'utiliser pleinement les propriétés du groupe symétrique pour le calcul polynomial à plusieurs variables et nous entendons continuer à développer ce point de vue.

Troisième Partie

Calcul Distribué en Combinatoire Algébrique

Introduction

Divide ut imperes.

Nous avons décidé en 1995 d'explorer la voie de la distribution de calculs en combinatoire, l'accès à une machine à architecture parallèle adaptée à nos besoins s'étant révélé trop coûteux. Plusieurs raisons ont guidé ce choix. D'abord l'investissement matériel et logiciel est réduit à zéro puisque l'objectif est d'utiliser au mieux des ressources existantes. De plus, le modèle de programmation par échange de messages au sein duquel nous développons les programmes distribués est bien adapté à nos préoccupations. Par exemple, nos programmes ne dépendent pas intrinsèquement du nombre des machines disponibles mais au contraire, au fur et à mesure de l'apparition de nouvelles machines à l'Université, nous espérons pouvoir les intégrer à la machine virtuelle afin que les programmes déjà écrits puissent tirer profit des nouvelles ressources.

Nous avons dès lors mis en place une machine virtuelle gérée par `PVM` (*Parallel Virtual Machine*) composée de 2 machines puis 3, et ainsi de suite jusqu'à disposer désormais de plus de vingt ordinateurs. Notre point de vue était simple : il fallait tout d'abord expérimenter pour appréhender la performance d'une telle machine, c'est-à-dire lancer des calculs pour apprécier la puissance globale de ce parc de machines hétérogènes regroupées de manière logicielle pour former la machine virtuelle. De manière générale, plusieurs axes expérimentaux ont été envisagés. Ils concernent notamment la mise en évidence des atouts mais aussi des limitations de cette machine virtuelle, ou encore la mise en avant d'une utilisation transparente pour l'utilisateur (programmeur) de l'ensemble des ressources. L'ensemble des expériences menées devrait à terme déboucher sur la réalisation d'un système de calcul symbolique distribué.

Nos premières expériences ont porté notamment sur le calcul distribué de π et la modification d'une bibliothèque de calcul sur entiers longs afin qu'elle intègre des composantes de distribution de calculs. Le premier exemple sert la difficile tâche d'évaluation des performances globales d'une machine virtuelle. Le deuxième exemple explore la voie de la transparence, transparence d'utilisation de la machine virtuelle s'entend : en d'autres termes, notre objectif a été d'étudier dans quelle mesure l'on pouvait modifier une librairie existante afin qu'elle utilise des ressources distribuées sans aucune modification pour le programmeur. Nous reviendrons brièvement sur ces deux exemples dans le chapitre 6.

Des expérimentations plus directement liées à la combinatoire ont été menées en collaboration avec l'Université de Bayreuth (Allemagne). Les équipes de combinatoire algébrique des universités de Marne-la-Vallée et Bayreuth étaient alors intéressées par la distribution de la résolution de problèmes combinatoires comme par exemple l'énumération et la classification des groupes résolubles à isomorphisme près ou encore la résolution du problème du sac à dos multi-dimensionnel. Plusieurs programmes ont été développés et testés sur les deux sites des universités de Marne-la-Vallée et Bayreuth. L'objectif était de cerner la puissance de calcul de la quarantaine de machines dont nous disposions.

Nous avons dans ce cadre mis au point un algorithme distribué de génération de l'ensemble des tableaux de rubans de forme et poids donnés. Cet algorithme permet aussi de calculer la fonction génératrice selon une statistique appelée *spin*. D'un autre côté, le programme permet de spécifier le nombre de processus s'exécutant simultanément sur la machine virtuelle. Dit autrement, nous pouvons volontairement limiter le nombre de tâches afin de simuler l'exécution parallèle de notre programme sur une machine intégrant un nombre fixé de processeurs. Il est clair que dans la plupart des cas le temps d'exécution d'un programme sur une machine parallèle va directement dépendre du nombre de processeurs, mais cette dépendance semble plus difficile à maîtriser dans le cas d'une machine virtuelle tant l'ensemble du calcul dépend de bien d'autres paramètres. Il n'est en fait pas facile, faute d'outils adaptés, d'évaluer les performances d'une machine virtuelle composée de machines hétérogènes. Nous avons essayé d'apporter une solution à ce problème très intéressant qui regroupe à la fois le domaine de la visualisation de l'exécution d'un programme parallèle et l'aide à la mise en évidence des performances d'un tel programme. Pour cela, nous avons modifié une surcouche logicielle de `PVM` afin de supprimer les limitations qui freinaient considérablement son utilisation. `PGPVM` est désormais capable de tracer l'exécution de n'importe quelle programme distribué à l'aide de `PVM` alors que la première version de Brad Topol n'autorisait que le traçage des programmes créant tous les processus (tâches) impliqués dans le calcul au début de celui-ci, et se terminant tous ensemble à la fin. Ce ne peut pas être le seul schéma possible d'exécution d'un programme distribué car `PVM` donne justement cette possibilité de création et de terminaison d'une tâche à n'importe quel moment. Nous détaillerons dans le chapitre 8 les modifications apportées et nous appliquerons ce nouvel outil au problème de l'énumération des tableaux de rubans (ce qui n'était pas possible avec la première version).

Le premier chapitre de cette partie donne une présentation succincte de la librairie `PVM` qui nous a permis de distribuer nos calculs. Nous présentons en particulier les deux exemples qui nous ont guidé dans l'étude des performances de la machine virtuelle et vers une utilisation de ses ressources transparente pour l'utilisateur.

Le deuxième chapitre présente un nouvel algorithme de calcul de l'ensemble des tableaux de rubans d'une forme et d'un poids donnés.

Cet algorithme qui produit aussi le polynôme de spin associé à cet ensemble de tableaux de rubans a donné lieu à une version parallèle effectivement réalisée sur une machine parallèle virtuelle. Le troisième chapitre présente cet algorithme et pose le problème de l'analyse de la performance des programmes distribués. Dans ce cadre, nous avons entrepris la modification d'un logiciel existant afin qu'il apporte une solution logicielle satisfaisante dans l'étude des performances d'un programme distribué à l'aide de la librairie `PVM`. Ce chapitre montre finalement les résultats obtenus sur l'algorithme de génération des tableaux de rubans.

6

Sur le calcul distribué

Où il n'y a point de maître, tout le monde est maître;
où tout le monde est maître, tout le monde est esclave.

— *Jacques Bénigne Bossuet.*

Un ordinateur parallèle est un ensemble de processeurs capables de coopérer afin de résoudre un problème. Cette définition plutôt vague a le mérite d'inclure les super-calculateurs parallèles à base de plusieurs centaines voire milliers de processeurs, les stations multi-processeurs ou encore les réseaux de stations de travail. Les ordinateurs parallèles sont intéressants en ce sens qu'ils offrent la possibilité de concentrer les ressources—aussi bien les processeurs, que la mémoire ou la bande passante d'entrée-sortie—sur un problème donné.

Bien que la puissance des ordinateurs soit en constante augmentation, les limites des machines sont bel et bien palpables. Munis d'un des plus puissants ordinateurs parallèles actuels, les météorologistes français ont bien du mal à calculer l'évolution de l'atmosphère terrestre pour établir une prévision satisfaisante à 48 heures. Les avions sillonnant sans cesse le ciel sont devenus leur principale source d'acquisition de données. Cela permet un suivi très précis de l'évolution de l'atmosphère et une prévision intéressante à court terme (quelques heures). Pourtant, la prévision du temps à plus long terme demande une puissance de calcul largement supérieure à celle proposée par les ordinateurs actuels et la théorie mise en jeu dans ces prévisions laisse plutôt place à de nouvelles techniques heuristiques afin de réduire les temps de calculs. En d'autres mots, alors que les ordinateurs deviennent de plus en plus rapides, il est aisé de se persuader qu'il existera toujours des problèmes ne pouvant pas être résolus par la technologie actuelle. A ce sujet, un rapport destiné au gouvernement britannique à la fin des années 40 estimait que les besoins en calculs de la Grande-Bretagne devraient potentiellement être couverts par deux voire éventuellement trois ordinateurs. Les auteurs de ce rapport ne considéraient que les applications des ordinateurs en science et ingénierie car ces ordinateurs étaient principalement utilisés pour des calculs balistiques, mais c'était sans compter les applications commerciales qui allaient sous peu inonder le marché du calcul. CRAY RESEARCH prédisait d'ailleurs à l'époque un marché de quelques dix super-calculateurs alors qu'en fait plusieurs centaines seront vendus. Bien que les applications commerciales influenceront grandement les architectures de la plupart des machines parallèles du futur, les applications scientifiques comme par exemple les simulations numériques de systèmes complexes comme le temps, le climat, les systèmes mécaniques, les circuits électroniques ou encore les réactions chimiques resteront des utilisateurs privilégiés de cette technologie des ordinateurs parallèles. La tendance claire qui semble se dessiner montre que les applications commerciales se confondent avec les applications plus scientifiques au niveau des ressources mises en jeu dans ces deux cadres : bien qu'initialement les programmes scientifiques demandaient plutôt du temps machine alors que les développements commerciaux étaient eux plutôt axés sur le traitement de quantités impressionnantes de données, les deux directions se rejoignent de plus en plus puisque ces dernières applications utilisent des algorithmes de plus en plus sophistiqués et dans le même temps, les logiciels scientifiques veulent traiter ou générer de plus en plus de données.

On peut voir cette évolution comme assez naturelle en ce sens qu'elle suit l'augmentation de la performance des machines. Pour les plus rapides des ordinateurs cette performance croit d'un facteur dix tous les cinq ans environ, cependant les architectures machine qui sont à la base de cette formidable croissance passent progressivement du séquentiel au parallèle. La performance d'une machine dépend directement du temps nécessaire à l'exécution d'une opération basique et du nombre de ces opérations élémentaires pouvant être réalisées simultanément. Le temps pour réaliser une opération basique est intrinsèquement limité par le "temps de cycle" du processeur. Mais voilà, les temps de cycles diminuent plutôt lentement et semblent se rapprocher des limites physiques imposées par la vitesse de la lumière. En d'autres mots, on ne pourra plus dépendre de processeurs plus rapides pour augmenter significativement les performances d'une machine donnée. Il faut donc s'orienter vers d'autres voies. Par exemple, les concepteurs d'ordinateurs favorisent la technique de "pipelining" dont l'objet est d'effectuer simultanément plusieurs étapes élémentaires pour certaines instructions, ou encore mettent en œuvre des unités de calculs dotées de plusieurs multiplieurs, additionneurs alimentés par un seul flot de données. D'autres constructeurs s'orientent vers l'intégration de plusieurs ordinateurs au sein d'une seule machine. Cet axe est largement favorisé par les avancées en matière de technologie de très haute intégration et le résultat fait que le nombre de processeurs par machine est en constante augmentation.

Une autre facette de cette évolution des performances est l'incroyable augmentation des capacités des réseaux d'interconnexions entre ordinateurs. Cela rend désormais possible ou tout au moins envisageable le développement d'applications utilisant concrètement des ressources distribuées comme si elles faisaient partie d'un même ordinateur. Il faut bien comprendre que le calcul sur un parc d'ordinateurs reliés par un réseau (calcul distribué) n'est pas une simple branche du calcul parallèle car il se heurte à des problèmes (comme par exemple la fiabilité, la sécurité et l'hétérogénéité des machines impliquées) parfois éloignés de ceux rencontrés ou envisagés au niveau du calcul parallèle. Pourtant les deux mondes se rapprochent quand même au niveau développement algorithmique puisque globalement l'enjeu est de mettre en place des programmes répartissant l'ensemble des tâches élémentaires à réaliser sur plusieurs machines à la fois.

En fait, la conception de programmes parallèles est profondément liée à un style particulier d'architecture parallèle (hypercube, tore, réseau rectangulaire, arbre, ...). L'algorithmique parallèle est donc étroitement dépendante de la nature même de la machine parallèle mais si cette dépendance entraîne parfois quelques difficultés de développement car le programme doit décrire précisément le calcul que chaque processeur doit effectuer, c'est vraisemblablement souvent "le prix à payer" pour obtenir des algorithmes performants. Un même algorithme peut éventuellement conduire à plusieurs versions parallèles. Ces versions dépendent essentiellement de la synchronisation des processeurs, du découpage en tâches élémentaires, de l'affectation des tâches aux différents processeurs ou encore tout simplement de l'accès aux données. D'un autre côté, l'approche "distribution de calculs" présente des avantages non négligeables. Comme la distribution c'est la répartition des tâches élémentaires qui réalisent un calcul global sur des ressources différentes *i.e.* essentiellement sur des machines différentes, chaque processeur a en charge une ou plusieurs tâches au sein d'un ensemble de tâches à effectuer. Par ailleurs, l'ensemble des processeurs—pour ne pas dire machines—est apte à communiquer avec n'importe quelle autre processeur au travers d'un réseau local (voire distant). Nous appellerons désormais cet ensemble de processeurs, *machine virtuelle* pour bien montrer que l'enjeu est de réaliser des programmes parallèles pour une machine parallèle au sens de multi-processeurs mais dont l'existence n'est que logicielle, c'est-à-dire simulée par l'abstraction logicielle de la réelle architecture de la machine alors que chacune de ses composantes est bien réelle.

D'une manière générale, le développement de programmes parallèles s'appuie sur des modèles de programmation particuliers. C'est bien entendu aussi le cas de la programmation des machines séquentielles, dont le modèle de von Neumann considère qu'un processeur est apte à exécuter des séquences d'instructions, chaque instruction pouvant spécifier en plus des opérations arithmétiques, les adresses des données à lire ou écrire voire l'adresse de la prochaine instruction à exécuter. Même s'il est toujours possible d'écrire des programmes de bas niveau sur la base de ce modèle (c'est-à-dire essentiellement en langage machine), cette méthode est de loin trop complexe car il faut organiser plusieurs milliers d'instructions et garder trace de l'emplacement mémoire de plusieurs millions de données par exemple. C'est pour cela que l'on préfère des techniques de conception modulaire où les instructions complexes sont construites à partir de composantes élémentaires, l'ensemble étant structuré en terme de structures de données, boucles d'itération ou encore procédures. Ce niveau d'abstraction a donné naissance à des langages dits de "haut niveau" comme le *Fortran*, le *Pascal* ou encore le *C*. Les programmes parallèles ne peuvent pas tout à fait s'appuyer sur ce type de modèle. Par exemple, dans le cadre d'une programmation bas niveau, s'ajoutent à la gestion des milliers d'instructions, l'administration des différents processeurs et la coordination globale du calcul. Le concept de modularité des programmes reste primordial mais de nouveaux concepts nécessitent d'être abstraits pour être intégrés au modèle de programmation : ce sont la concurrence entre tâches, la capacité d'adaptation d'un programme en fonction du nombre de processeurs et la localité de certaines données.

Un modèle semble se détacher en mettant en avant les deux notions de tâche et de canal de communication. Ce modèle donne aux tâches la possibilité d'effectuer quatre actions principales : envoyer un message, recevoir un message, créer de nouvelles tâches et se terminer. De nombreux autres modèles ont suscité de l'intérêt comme par exemple le modèle à base d'échange de messages dans lequel les programmes peuvent créer plusieurs tâches, chaque tâche, identifiée par un label (nom ou numéro) unique, pouvant interagir avec les autres en s'envoyant des messages. De manière générale, lorsque l'algorithmique parallèle est plongée dans un modèle donné, elle acquiert de l'indépendance par rapport au support physique, qu'il soit encore une fois une simple machine ou un réseau de machines. Comment un tel modèle peut-il être réalisé ? Disons qu'il existe deux architectures principales, l'une étant appelée *SIMD* (*single instruction, multiple data*) et l'autre *MIMD* (*multiple instructions, multiple data*).

La technologie des "machines massivement parallèles" donne accès à un nombre important de processeurs interconnectés de différentes manières comme par exemple en un tableau de processeurs, un arbre binaire ou encore un hypercube. Les processeurs peuvent communiquer très rapidement avec chacun de leurs voisins et il y a aussi la possibilité d'un échange global de données, par exemple avec la machine frontale. Une communication globale entre processeurs est possible mais la communication locale est toujours plus rapide. L'architecture même a une autre limitation claire essentiellement contenue dans son nom : il y a toujours une seule instruction exécutée simultanément par l'ensemble des processeurs. Le concept de parallélisme tient au fait que l'on laisse chaque processeur travailler sur ses propres données. Ces machines peuvent être programmées avec un langage de haut niveau comme par exemple les machines MasPar qui disposent d'un compilateur proche du compilateur *C* mais qui l'étend en introduisant des variables et instructions de type *plural*. Une variable *plural* existe sur chaque processeur (chaque processeur ayant une instance particulière de la variable) et les instructions de type *plural* s'exécutent seulement sur l'ensemble des processeurs actifs qui par définition est l'ensemble des processeurs allant prendre part à la prochaine instruction.

Le programmeur de machines séquentielles trouvera intéressant l'exemple de l'instruction conditionnelle `if then else` exécutée sur ses deux branchements possibles. L'expression de contrôle (`if`) est en fait utilisée pour modifier l'ensemble des processeurs actifs. Seuls les processeurs (parmi les processeurs actifs) ayant évalué l'expression de contrôle à la valeur vrai exécuteront la première branche (`then`) de l'instruction. L'ensemble complémentaire des processeurs pris parmi les processeurs actifs à l'entrée dans l'instruction `if then else` exécutera le second branchement (`else`) le cas échéant (s'il y en a un). Il y a aussi des instructions `while` et `for` de type *plural* qui ont ce même effet de modification de l'ensemble des processeurs actifs à un instant donné. Il est clair que de nombreuses applications peuvent tirer profit de cette architecture mais c'est au prix d'une modification profonde des algorithmes impliqués et d'une utilisation des langages spécifiques dédiés à ce type de machines, pour ne pas dire propre à tel ou tel type d'ordinateur massivement parallèle.

En contraste avec le concept de *SIMD* où chaque processeur n'est parfois ni très puissant ni même dotée d'une mémoire confortable, l'architecture *MIMD* affiche d'autres ambitions : chaque processeur correspond souvent à une vraie station de travail avec beaucoup de mémoire, dotée d'un disque local. Connectés par un réseau, ces processeurs travaillent indépendamment les uns des autres, les communications étant le plus souvent asynchrones. Chaque entité de calcul peut accueillir plusieurs tâches et le réseau d'interconnexion permet une communication entre deux processeurs quelconques de l'architecture. Le calcul sur un réseau hétérogène a plusieurs avantages : par exemple le coût des calculs peut être très faible puisque l'on utilise des ressources déjà existantes et, d'un autre côté, la performance peut être optimisée en associant chaque tâche à l'architecture la plus appropriée. Par ailleurs, la programmation concurrente sur un réseau de stations offre un avantage certain en matière de développement car les outils d'édition, de compilation et de mise au point sont très souvent déjà disponibles. Un autre attrait de ce type de programmation est le potentiel de gestion des erreurs qui est laissé au programmeur. Celui-ci a la possibilité d'associer à son programme un niveau de tolérance à l'erreur qui peut dans certains cas lui permettre de ne pas complètement arrêter un calcul dès lors qu'une erreur logicielle ou matérielle est survenue. Toutes ensemble, les stations de travail constituent une "machine virtuelle", notion qui nous amène tout naturellement à parler de PVM.

PVM est une librairie de fonctions en *C* ou *Fortran* dédiée à la distribution de calculs, associée à un démon. PVM n'est certes pas le seul logiciel à offrir cette possibilité de distribution de calculs mais il semble tout de même s'imposer. Sa disponibilité en standard sur les machines CRAY, sa simplicité d'utilisation, mais aussi la dynamique de développement installée autour du concept même ont en fait guidé notre choix initial.

Le démon pvmd3 doit être installé sur toutes les machines qui composent la machine virtuelle. Il peut être installé par n'importe quel utilisateur. Lorsqu'un utilisateur souhaite lancer un programme il doit auparavant exécuter pvmd3 sur une des machines qui a son tour lancera les démons de tous les autres ordinateurs constituant la machine virtuelle.

La librairie PVM permet à un ensemble de machines hétérogènes d'être vue comme une simple machine parallèle virtuelle. L'interface de programmation autorise la création et la terminaison de tâches et fournit des fonctions de communications entre tâches. PVM prend en compte tous les problèmes de routage de messages, de conversion de données, de répartitions des tâches au sein d'un réseau de machines à architectures parfois incompatibles. PVM utilise les réseaux d'interconnexions existants et ne nécessite pas de structure particulière (en anneau ou en étoile par exemple). La librairie est disponible sur de nombreuses plate-formes UNIX, ordinateurs massivement parallèles et autres ordinateurs vectoriels avec des portages sous MS-WINDOWS et OS/2. Les programmes compilés sur chacune des différentes architectures peuvent communiquer entre eux, cette communication étant orchestrée par les démons PVM tournant sur chaque composante (ordinateur) de la machine virtuelle. Cette dernière est construite par l'utilisateur qui décide des machines à inclure grâce à un fichier ASCII qui contient des informations relatives à chacune d'elles.

La structure atomique d'un programme PVM est la *tâche* (essentiellement un processus UNIX) : flot indépendant d'instructions séquentielles qui oscille entre communication et calcul. PVM n'impose pas de correspondance processus-processeur mais autorise au contraire que plusieurs tâches soient potentiellement prises en charge par un seul processeur. Chaque tâche est identifiée par un numéro unique au sein de la machine virtuelle.

Côté utilisateur, le tout peut être vu comme un ensemble de processeurs virtuels sans propriété particulière ou alors comme un ensemble d'unités de calculs ou chaque unité exécutera la tâche la plus appropriée par rapport aux ressources dont elle dispose à un instant donné. Le programmeur divise ainsi son algorithme en plusieurs tâches qu'il gère explicitement, c'est-à-dire qu'il crée ou termine. Il prend aussi en compte la communication entre les tâches dans un modèle où chaque tâche peut échanger de l'information avec n'importe quelle autre tâche. A sa disposition, la console PVM plutôt rébarbative et peu conviviale, mais aussi l'interface graphique XPVM offrant en fin de compte presque le strict nécessaire à une utilisation facile du concept de machine virtuelle : possibilité d'ajout ou de retrait d'une machine, visualisation des tâches par machine mettant en évidence les trois états possibles d'activité (calculatoire, en attente d'un message, en opération de communication différente de la réception d'un message) et des échanges de messages, visualisation du pourcentage de chacun de ces 3 états sur l'ensemble des tâches en fonction du temps et tout récemment, visualisation des files de messages associées à chaque tâche.

En ce qui concerne l'utilisation même de PVM, ce sont à peu près les seuls outils disponibles sur le marché "*Internet*", d'autres logiciels sont à l'état de premières versions peu ou pas documentées ce qui ne facilite pas leur utilisation. L'objet de XPVM n'est pas vraiment de fournir un outil d'analyse de performance mais plutôt de donner à l'utilisateur un certain confort dû en grande partie à la visualisation de l'exécution même, et pas forcément à cause des quelques fenêtres d'informations sur le calcul.

Nous verrons dans le chapitre 8 que le programmeur dispose par contre d'outils pour la visualisation et l'analyse des performances d'exécution d'un programme distribué à l'aide de `PVM`. Nous y expliquons notamment comment et pourquoi nous avons modifié une surcouche de `PVM` (`PGPVM`) afin qu'elle offre globalement un service de traçage d'exécution digne de ce nom dans le contexte du calcul distribué.

Notre étude des possibilités d'utilisation de `PVM` a conduit à la programmation de deux exemples dont l'objet premier a été la mise en évidence des possibilités, limitations et performances de la machine virtuelle. Ceci nous a permis de faire le point sur les futurs axes de développement en matière de calcul distribué, en précisant notamment la nature des calculs dont la distribution sur un parc de machines apportera beaucoup, mais aussi sur le degré de transparence que l'on peut espérer au sujet de l'utilisation de la machine parallèle virtuelle.

6.1 Calcul distribué de la constante π

Considérant un cercle de rayon 1, nous savons que son aire est $\pi = 3.14159265\ldots$, les points signifiant que les chiffres ne s'arrêtent pas mais continuent à l'infini. Ces chiffres du développement décimal de π ont fasciné de nombreuses personnes depuis des siècles. Les Babyloniens étaient arrivés à une valeur de $\pi = 3 + 1/8$ et les Egyptiens à la valeur de $\pi = (16/9)^2$. Archimède établit que :

$$3 + \frac{10}{71} < \pi < 3 + \frac{1}{7} \quad ,$$

et $\pi \approx 211875/67441 = 3.14163$. Appolonius améliora cette approximation et Ptolémée au second siècle utilisa $\pi = 377/120 = 3.14166$. Au cinquième siècle, Tsu Chung-Chi établit que :

$$3.1415926 < \pi < 3.1415927 \quad .$$

Al-Kashi de Samarkand, avant 1436, donna les 14 premières décimales de π et en 1593, c'est Adriaen van Roomen qui apporta la 15-ième. Trois ans après, Ludolph van Ceulen calcula les 32 puis les 35 premières décimales. Et puis Sharp (1705) donna 72 décimales, Machin (1706) 100 décimales, De Lagny (1719) 127 décimales, Vega (1794) 140 décimales, Strassnitzky et Dase (1844) 200 décimales, Richter (1855) 500 décimales, Shanks (1873) 707 décimales (en 1945, Ferguson découvrit que le calcul de Shanks était faux à partir de la position 527...), Ferguson (1946) 620 décimales. En 1947, Ferguson donna 808 décimales à l'aide d'un petit calculateur et en 1949 l'ENIAC était programmé pour produire 2037 décimales. Le NORC a produit à son tour en 1954 3089 décimales et en 1957 le PEGASUS (Londres) en calcula 7480. En 1959 à Paris, un IBM 704 a fourni les 16167 décimales de π et en 1961 Shanks et Wrench améliorèrent un programme qui donna à New York sur un IBM 7090 les 100000 premières décimales. En 1966, le record fut porté à Paris à 250000 décimales sur un IBM 7030 puis à 500000 décimales sur un CDC 6600.

Bien qu'il n'y ait pas vraiment d'application pratique à la connaissance de plusieurs milliards de chiffres après la virgule de π, on tente quand même de les calculer pour le plaisir, ou pour le record qui est depuis le 24 septembre 1995 installé à 6442450000 de chiffres décimaux.

Voici une manière très lente de calculer π. Inscrivons un cercle de rayon 1 dans un carré de côté 2. Suspendons le total sur un mur et lançons complètement aléatoirement des fléchettes dans le carré. Supposons que sur les N fléchettes qui heurtent l'intérieur du carré, M d'entre elles se situent à l'intérieur du cercle. Les lois de la probabilité nous disent qu'il est raisonnable d'espérer que M/N est approximativement égal à $\pi/4$ car l'aire du cercle vaut π et celle du carré vaut 4. Des années durant, on a recherché des algorithmes pour calculer π, plus efficaces que l'algorithme décrit ci-dessus. Cela donna lieu à de nombreux résultats parfois mathématiquement très sophistiqués et souvent compliqués à programmer sur un ordinateur[(1)].

Maintenant, si nous changeons un peu la question pour désormais nous demander quelle peut être la i-ième décimale de π plutôt que de chercher à les calculer toutes, le problème est tout autre. Jusqu'à présent, on croyait que calculer la i-ième décimale d'un nombre transcendant comme π était aussi difficile que de calculer π lui-même. En fait, il n'en est rien comme viennent de le montrer David Bailey, Peter Borwein et Simon Plouffe [2]. Ils viennent en effet de calculer le 10 milliardième chiffre du développement hexadécimal de π[(2)] en utilisant la formule :

$$\pi \quad = \quad \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n \quad . \tag{6.1}$$

Cette formule permet de calculer la i-ième “hexadécimale” de π sans être obligé de calculer toutes les précédentes. Personne n'avait auparavant seulement conjecturé l'existence d'un tel algorithme. D'un autre côté, la question suivante reste évidemment en suspens : existe-t-il des polynômes $a(n)$ et $b(n)$ à coefficients entiers tels que :

$$\pi \quad = \quad \sum_{n=0}^{\infty} \frac{a(n)}{b(n)} \left(\frac{1}{10} \right)^n \quad ?$$

Selon David Bailey, Peter Borwein et Simon Plouffe, partant de cette formule, on peut déduire un algorithme ne nécessitant pas d'arithmétique multi-précision, n'utilisant que peu de mémoire et dont les performances seraient linéaires par rapport à l'ordre de la décimale désirée, pour calculer le développement hexadécimal de π. Ils ont utilisé leur formule dans un programme *Fortran* pour trouver cette 10 milliardième décimale de π sur un ordinateur très rapide de la NASA. Cela a pris seulement 96 heures pour réaliser ce travail qui constitue d'ores et déjà un nouveau record en la matière. Le point fort n'est pas tellement ce nouveau record mais la méthode utilisée.

Nous nous intéressons *a priori* à de plus modestes résultats. L'objet de l'exemple que nous présentons est de distribuer sur plusieurs machines le calcul des décimales de π à l'aide de `PVM`. Nous avons mis au point en collaboration avec Laurent Lheur, dans un premier

---

[(1)] Par exemple, MAPLE (depuis la version V) calcule π par la formule de Chudnovsky :

$$\frac{1}{\pi} = \frac{6541681608}{640320^{3/2}} \sum_{k=0}^{\infty} \left(\frac{13591409}{545140134} + k \right) \frac{(6k)!}{(3k)!(k!)^3} \frac{(-1)^k}{640320^{3k}} \quad .$$

[(2)] Ce chiffre est égal à 9.

temps une version séquentielle (écrite en langage C) de l'algorithme de calcul d'une tranche de chiffres hexadécimaux de π à partir d'une position donnée. A titre purement indicatif, les temps de calculs s'échelonnent pour la position 10^6 entre 76 secondes pour un HP-9000 755 à 99Mhz, 102 secondes pour un PC Pentium 100Mhz, 125 secondes pour une Silicon Graphics de type Onyx et 240 secondes, pour une Silicon Graphics de type Indy. Nous avons ainsi pu vérifier, comme annoncé, que le temps calcul d'une tranche d'une dizaine de chiffres hexadécimaux à partir de la position p était en $\mathcal{O}(p)$.

Dès lors que les temps de calculs séquentiels nous ont semblé satisfaisants, nous avons entrepris la distribution du calcul des tranches. Le modèle maître/esclave a été retenu avec un programme maître administrant le calcul global et des esclaves lancés sur les différentes machines calculant les tranches demandées. Encore une fois, à titre d'information, nous avons calculé en parallèle en à peu près 40 minutes les 100000 premières hexadécimales de π alors qu'il nous a fallu plus de 10 heures en séquentiel.

Ce petit exemple a eu l'intérêt de mettre à jour un des gros problèmes de la distribution de calculs sur un parc de machines hétérogènes. La répartition des tâches doit absolument prendre en compte la puissance des ordinateurs et pas seulement la charge de la machine à un instant donné. `PVM` inclut un algorithme de répartition en fonction de la charge de la machine mais il n'offre pas cette possibilité de placement des tâches en fonction d'un indice de performance de la machine par exemple(3).

En fin de compte, c'est à l'utilisateur de répartir ses calculs en fonction des performances des différentes machines. Cela pose un réel problème car l'utilisateur a tout avantage à ignorer les caractéristiques des processeurs qui composent la machine virtuelle et on peut même imaginer qu'il ne sache pas exactement combien de machines composent la machine virtuelle. Par ailleurs, la prise en charge de la répartition des tâches ne peut que compliquer l'algorithme à distribuer.

6.2 Utilisation transparente de ressources

Nous avons précisé dans l'introduction qu'un de nos objectifs était de fournir à l'utilisateur un ensemble de ressources utilisable le plus simplement possible. Notre point de vue n'est donc pas tellement dans ce cadre de créer et d'exploiter une machine parallèle virtuelle ayant telle ou telle caractéristique technique mais plutôt de regarder ce que peut apporter un ensemble de processeurs travaillant en concurrence pour réaliser un travail global.

Parmi les nombreuses possibilités d'expérimentation, nous avons retenu celle de l'utilisation d'une bibliothèque de fonctions mathématiques. La librairie `LIP` (*Long Integer Package*) développée par Arjen K. Lenstra [8] propose un ensemble de fonctions permettant de calculer sur entiers très longs en donnant notamment les opérations arithmétiques (modulaires) usuelles sur entiers mais aussi en offrant des fonctionnalités plus évoluées comme par exemple la factorisation par la méthode des courbes elliptiques. Parmi l'ensemble des fonctions nous en avons retenu 6 dont le calcul distribué présentait un intérêt pratique. Nous avons entrepris, avec l'aide de Laurent Lheur, la modification de la bibliothèque afin qu'elle intègre aux endroits les plus favorables des ordres de distribution

(3)En fait, cette option est prévue puisque la description de la machine virtuelle inclut un champ permettant de spécifier par un entier la performance relative des machines. Pourtant, la documentation précise que le placement des tâches suit actuellement une heuristique mais utilisera la charge des machines et l'indice de performance dans une version future afin de déterminer le meilleur hôte à un instant donné.

de calculs. Le résultat a été la mise en place d'une librairie `LIP_PVM` offrant exactement les mêmes fonctionnalités que la librairie initiale `LIP` mais donnant en plus à l'utilisateur des possibilités d'utilisation des ressources d'une machine parallèle virtuelle de manière transparente, c'est-à-dire sans aucune modification de ses programmes.

Les fonctions intégrant ces nouvelles possibilités sont essentiellement probabilistes et notre objectif était très simplement d'augmenter la probabilité de succès et de diminuer les temps d'exécution de ces algorithmes. Nous nous sommes notamment intéressés à la fonction `zecm` dont l'objet est de trouver un facteur premier d'un entier très long n par la méthode des courbes elliptiques. Cette fonction est vraisemblablement un des fleurons de cette librairie. Cette méthode originale dépend de nombreux paramètres qui influent sur la probabilité de succès des différents essais entrepris.

✍**Remarque**. La documentation précise d'ailleurs à ce sujet quelques choix optimaux de paramètres en fonction d'une estimation du nombre de chiffres décimaux du plus petit facteur premier de n, pour obtenir une probabilité de succès de 60%.

Entiers à factoriser	`LIP`	`LIP_PVM`
7×479001599	✓	✓
23×49999	✓	✓
719×1231101931	✓	✓
49999×39916801	✓	✓
1319963×87178291199	✓	✓
479001599×87178291199	✓	✓
87178291199×1023456987896543201	✗	✓
808000000823×105105000105167	✗	✓
8808000000883×1023456987896543201	✗	✓
42842283925351×1023456987896543201	✓	✓
4953823561938703×837413333000293843	✗	✓
837413333000293843×33384733910573629399	✓	✓
33384733910573629399×333934728274373773223489	✗	✓
10398483827437377322239×333934728274373773223489	✓	✓

Figure 6.1: *Comparaison des bibliothèques* `LIP` *et* `LIP_PVM`.

Chaque essai est divisé en deux phases. La première choisit une courbe elliptique aléatoire C et un point x sur C et calcule x^k sur C pour un certain k. Si la courbe C est chanceuse, le calcul de x^k va échouer et un facteur de n est détecté, dans le cas contraire il faut passer à la deuxième phase. La deuxième phase se charge de comparer différentes puissances de x^k et si C est bien choisie alors un facteur de n sera détecté à la fin de cette phase. Si aucun facteur n'est trouvé, on réitère ce même algorithme sur une courbe différente. C'est en fait à ce niveau précis qu'intervient la distribution du calcul car l'appel à la fonction `zecm` va désormais dans la nouvelle librairie lancer plusieurs essais sur plusieurs courbes en parallèle, augmentant ainsi la probabilité de succès. Côté utilisateur, le même appel à la fonction `zecm` sera fait, son programme ayant été compilé en utilisant la nouvelle librairie. S'il positionne à la compilation l'option `-DUSE_LIBPVM`, les ressources de la machine parallèle virtuelle seront mises à contribution et dans le cas contraire l'exécution de la fonction sera séquentielle.

La table 6.1 donne des exemples de factorisation pour lesquels nous avons successivement utilisé la librairie `LIP` puis la librairie `LIP_PVM`. Elle indique pour quelques produits de deux nombres premiers[(4)] si la factorisation a réussi (symbole ✓) ou échoué (✗) respectivement sans `PVM` et avec `PVM`.

Nous devons ajouter que de nombreux tests ont été réalisés en dehors des résultats présentés et il nous faut constater qu'un facteur a toujours été trouvé lorsque nous utilisions notre nouvelle bibliothèque alors que de nombreuses fois, la version séquentielle de la fonction `zecm` n'a pas été capable de produire un facteur.

Cela ne s'arrête pas là puisque les temps mis pour trouver un facteur composant un entier ont été bouleversés, expérimentalement dans le bon sens. En effet, les différents essais de courbes elliptiques effectués en parallèle avec la nouvelle librairie conduisent parfois à l'investigation de meilleures courbes, meilleures signifiant produisant simplement ou plus efficacement un des facteurs composant l'entier de départ. La table ci-après résume quelques résultats en fonction du nombre de chiffres décimaux du facteur le plus petit composant l'entier à factoriser. Les temps de factorisation sont donnés en secondes :

Nombre de chiffres	`LIP`	`LIP_PVM`	**Nombre de chiffres**	`LIP`	`LIP_PVM`
1	**0.19**	0.25	12	9.18	**6.30**
2	**0.02**	0.27	14	**1.66**	**1.66**
3	**0.00**	0.28	15	1.28	**0.74**
5	**0.01**	0.35	16	86.50	**10.62**
7	**0.03**	0.38	18	**72.54**	86.26
9	2.55	**0.54**	20	1204.29	**45.49**
10	0.79	**0.68**	22	1381.59	**134.12**
11	11.79	**1.6**	24	stoppé	**78.95**

Les temps qui sont intéressants concernent des entiers ayant un plus petit facteur de plus de 15 chiffres environ. Par exemple, le facteur à 22 chiffres a été détecté au bout de plus de 20 minutes sans `PVM` alors qu'avec `PVM` seulement 45 secondes ont été nécessaires. De même

(4) Le paramètre important est le nombre de chiffres décimaux du plus petit des deux facteurs premiers composant l'entier à factoriser.

pour 24 chiffres, un peu plus d'une minute a suffit avec `PVM` alors que sans `PVM` l'exécution a été interrompue après 2 heures de calculs !

Afin d'apprécier les performances de la librairie `LIP_PVM`, nous avons mis au point un programme se chargeant de factoriser complètement un entier en facteurs premiers. Pour cela, une tâche `PVM` maître `ecm_master` gère les différents facteurs qui lui parviennent des nb tâches esclaves `ecm_slave` chargées de trouver des facteurs à l'aide de la fonction `zecm`. Mais comme nous utilisons notre librairie les nb tâches lancent à leur tour plusieurs tâches `PVM`, le tout essayant donc l'algorithme sur de nombreuses courbes elliptiques simultanément. Dès qu'un facteur parvient à la tâche maître, celle-ci le traite afin de l'intégrer ou non à la liste des facteurs déjà trouvés. Notamment, si l'entier de départ n'est pas divisible par le produit de tous les facteurs trouvés y compris le nouvel arrivant, alors ce dernier sera oublié. Par contre, dans le cas contraire, si le facteur est premier, il sera directement intégré à la liste et sinon la tâche maître le stockera en attendant de pouvoir relancer des tâches qui le factoriseront. Lorsqu'il n'y a plus d'entiers à traiter l'algorithme s'arrête et produit la liste des facteurs premiers composant l'entier de départ.

En résumé, `PVM` est une librairie de fonctions en *C* ou *Fortran* dédiée à la distribution de calculs. Sa disponibilité en standard sur les machines CRAY mais surtout sa simplicité d'installation et d'utilisation ont fait que nous avons choisi ce système pour distribuer nos calculs. Nous avons alors entrepris une étude pragmatique des possibilités d'utilisation de `PVM` basée notamment sur le développement de deux exemples ayant pour objectif la mise en évidence des possibilités, limitations et performances d'une machine parallèle virtuelle.

Références bibliographiques

[1] S.G. Akl. *The design and analysis of parallel algorithms.* Prentice-Hall International Editions, (1989).

[2] D. Bailey, P. Borwein, and S. Plouffe. On the rapid computation of various polylogarithmic constants. (1995).

[3] J.-P. Banâtre. *La programmation parallèle.* Eyrolles, (1991).

[4] P. Beckmann. *A history of PI.* Barnes & Noble, (1971).

[5] A. Betten, S. Veigneau, and A. Wasserman. Parallel computation in Combinatorics using PVM. *Electr. J. of Discr. Math. and Theor. Comp. Sci.*, (1996). soumis.

[6] M. Cosnard and D. Trystan. *Algorithmes et architectures parallèles.* InterEditions, (1993).

[7] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms.* The Benjamin/Cummings Publishing Company, (1994).

[8] A.K. Lenstra. *Documentation of LIP*, (1995).

[9] T.G. Lewis and H. El-Rewini. *Introduction to parallel computing.* Prentice-Hall International Editions, (1992).

7

Enumération de tableaux de rubans et calcul du polynôme de spin

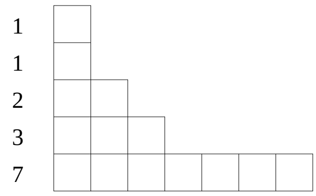
Réservant au peintre la tâche sévère et contrôlable
de commencer les tableaux,
attribuons au spectateur le rôle avantageux,
commode et gentiment comique
de les achever par sa méditation ou son rêve.

Ne serait-ce que par le nombre de leurs interprétations [7], les tableaux de Young sont des objets fondamentaux de la combinatoire algébrique. Ils représentent notamment les bases des représentations irréductibles du groupe symétrique et du groupe linéaire, des idempotents dans l'algèbre du groupe symétrique, les monômes dans le développement d'une fonction de Schur, des mots donnant naissance au monoïde plaxique, des décompositions réduites des éléments du groupe symétrique (monoïde nilplaxique). Ils ont aussi des interprétations géométriques, comme monômes en les coordonnées de Plücker pour la grassmannienne ou la variété de drapeaux (en langage moderne, ce sont des bases de sections de fibrés inversibles au dessus des sous-variétés de Schubert de la variété de drapeaux). Dans ce chapitre, nous nous intéressons à des généralisations des tableaux de Young qui ont déjà trouvé plusieurs applications intéressantes : les *tableaux de rubans*.

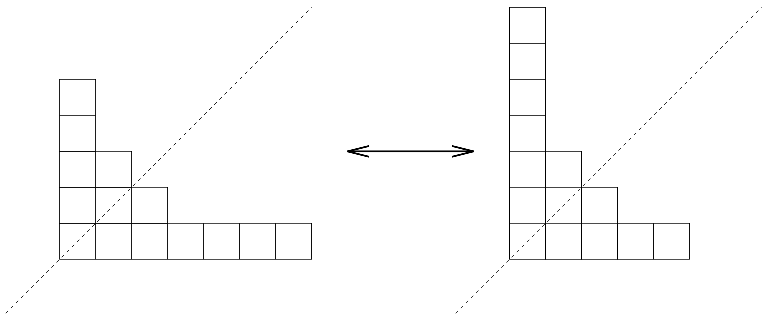
Rappelons qu'une partition $\lambda = (\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0)$ est une séquence décroissante finie d'entiers positifs λ_i appelés parts de la partition λ. La somme de toutes les parts λ_i est appelé le poids de λ et noté $|\lambda|$(1). La *longueur* $l(\lambda)$ de la partition λ est égale au nombre de parts de λ.

Comme nous l'avons déjà précisé, on peut représenter graphiquement une partition au moyen d'un diagramme de boîtes dans le plan, appelé diagramme de Ferrers : les lignes de ce diagramme sont numérotées de bas en haut, la i-ième ligne contenant exactement λ_i boîtes. Par exemple, à la partition $\lambda = (7, 3, 2, 1, 1)$ correspond le diagramme de Ferrers suivant :

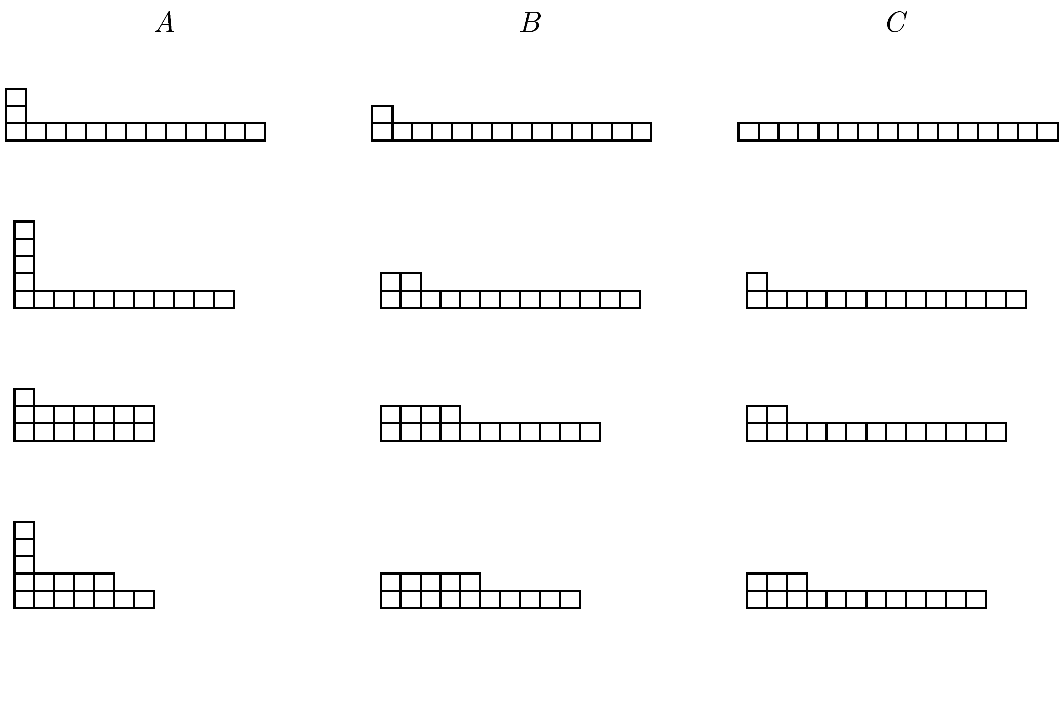
(1) Lorsque l'on parle de partition de n, on fait référence à une partition de poids n.

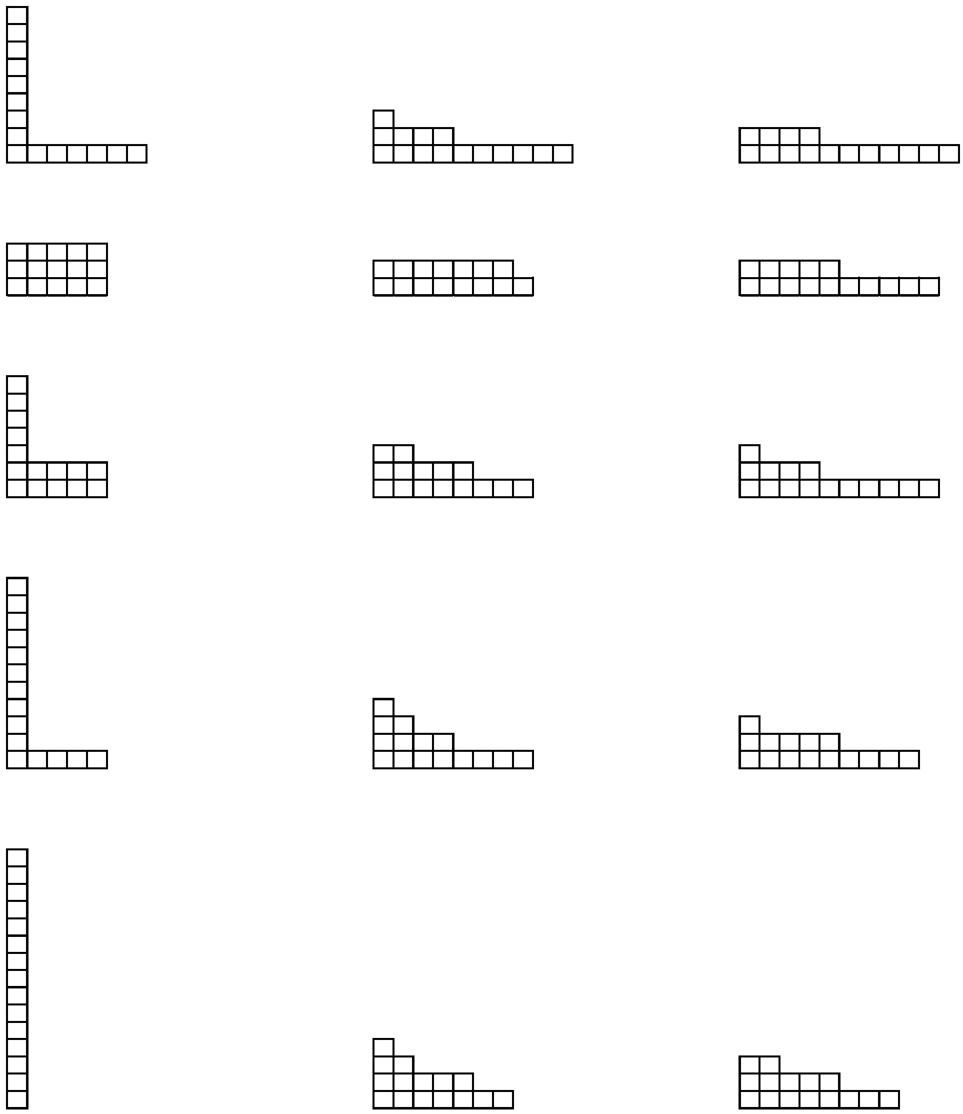


Maintenant, en renversant le diagramme suivant sa diagonale principale on préfère lire les nombres de boîtes dans les colonnes plutôt que dans les lignes. Cette opération graphiquement très simple associe à toute partition λ sa partition *conjuguée* λ'. Dans l'exemple précédent, la partition conjuguée de $\lambda = (7,3,2,1,1)$ est $\lambda' = (5,3,2,1,1,1,1)$:



Le théorème de Schur (1926) montre par exemple que le nombre $A(n)$ de partitions de n en parts congrues à plus ou moins 1 modulo 6 est égal au nombre $B(n)$ de partitions de n en parts distinctes congrues à plus ou moins 1 modulo 3 mais en plus est égal au nombre $C(n)$ de partitions de n dont les parts diffèrent d'au moins 3 et dont les parts multiples de 3 diffèrent au moins de 6. L'exemple suivant illustre cette propriété pour $n = 15$ en exhibant les 27 diagrammes de Ferrers représentant les partitions solutions correspondant à chacun des nombres $A(15) = B(15) = C(15) = 9$:





Si $p(n)$ désigne le nombre de partitions de l'entier n en parts quelconques, on a $p(0) = 1$, $p(1) = 1$, $p(2) = 2$, $p(3) = 3$, $p(4) = 5$, $\ldots$, $p(10) = 42$, $\ldots$, $p(200) = 3972999029388$, $\ldots$, $p(1000) = 24061467864032622473692149727991$. En fait, il existe une formule de récurrence très rapide pour calculer ces nombres :

$$p(n) = \sum_{k\geq 1} (-1)^{k-1} \left(p\Big(n - \frac{k(3k-1)}{2}\Big) + p\Big(n - \frac{k(3k+1)}{2}\Big) \right) \quad . \tag{7.1}$$

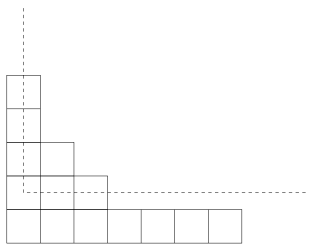
Un *tableau de Young* [4] ou tout simplement tableau, est le remplissage d'un diagramme de Ferrers correspondant à une partition donnée λ. On inscrit dans chaque boîte un entier positif de telle sorte que la lecture de gauche à droite de chaque ligne du diagramme soit une suite croissante au sens large et que la lecture de chaque colonne produise une suite croissante au sens strict cette fois. La partition λ est alors appelée la *forme* du tableau et si l'on utilise μ_1 fois le chiffre 1, μ_2 fois le chiffre 2, et ainsi de suite pour effectuer le remplissage, on dira que le *poids* ou l'*évaluation* du tableau est $\mu := (\mu_1, \mu_2, \ldots)$. Par exemple :

$$\begin{array}{|c|c|c|c|c|c|c|}
\cline{1-1} 8 \\
\cline{1-1} 6 \\
\cline{1-2} 3 & 5 \\
\cline{1-3} 2 & 2 & 6 \\
\hline 1 & 1 & 1 & 2 & 4 & 4 & 7 \\
\hline
\end{array}$$

est un tableau de forme $\lambda = (7, 3, 2, 1, 1)$ et de poids $\mu = (3, 3, 1, 2, 1, 2, 1, 1)$. Encore une fois, les résultats obtenus sur les tableaux montrent l'intérêt probant que les combinatoristes portent à ces structures très simples. Un des plus beaux exemples est sans doute la formule des équerres qui donne le nombre de tableaux standards f_λ d'une forme donnée λ. Un tableau est dit *standard* lorsqu'il est rempli avec des entiers tous distincts ou en d'autres termes lorsque son poids est le vecteur $(1, 1, 1, 1, \ldots)$. Ce nombre de tableaux standards d'une forme donnée λ est fondamental en théorie des représentations des groupes puisqu'il donne la dimension $\dim(V_\lambda)$ de la représentation irréductible du groupe symétrique correspondant à la partition λ :

$$f_\lambda = \dim(V_\lambda) := \frac{|\lambda|!}{\prod h_i} \quad , \tag{7.2}$$

où les h_i sont les longueurs des équerres du diagramme de Ferrers associé à la partition λ. Pour chaque boîte b du diagramme, la longueur de l'équerre associée à b est le nombre de boîtes sur la même ligne à droite de b et sur la même colonne au-dessus de b en comptant b une fois. L'exemple suivant montre en pointillé une équerre de longueur 6 :



La formule des équerres nous précise ainsi par exemple que le nombre de tableaux standards de forme $\lambda = (7, 3, 2, 1, 1)$ est égal à :

$$f_{(7,3,2,1,1)} = \frac{14!}{1 \times 2 \times 1 \times 4 \times 1 \times 3 \times 6 \times 11 \times 8 \times 6 \times 4 \times 3 \times 2 \times 1} = 47775 \quad .$$

On peut coder un tableau de Young de forme λ et de poids μ par une matrice de 0 et 1 ayant $l(\mu)$ lignes et $\lambda_1 = l(\lambda')$ colonnes. La i-ième ligne de cette matrice (en comptant de bas en haut) décrit les positions des boîtes numérotées i dans le tableau en mettant un 1 dans la colonne c de la matrice telles qu'il existe une boîte numérotée i dans la colonne c du tableau de Young. Reprenons l'exemple précédent du tableau de forme $(7, 3, 2, 1, 1)$ et de poids $(3, 3, 1, 2, 1, 2, 1, 1)$ dont la matrice associée a 8 lignes et 7 colonnes :

$$
\begin{array}{ccccccc}
8 & & & & & & \\
6 & & & & & & \\
3 & 5 & & & & & \\
2 & 2 & 6 & & & & \\
1 & 1 & 1 & 2 & 4 & 4 & 7
\end{array}
\qquad
\begin{array}{ccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 2 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 3 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 3 \\
\hline
5 & 3 & 2 & 1 & 1 & 1 & 1 &
\end{array}
$$

En comptant le nombre de 1 par ligne on retrouve évidemment le poids du tableau et en comptant le nombre de 1 par colonne on déduit la forme conjuguée du tableau, comme indiqué en dernière colonne et dernière ligne de la précédente matrice.

Toutes les matrices de 0 et de 1 ne codent pas des tableaux. Une matrice est un tableau de Young si on ajoute la condition que les sommes cumulées composante à composante des lignes de la matrice (lue de bas en haut) sont des partitions. On se persuade sans peine que les sommes cumulées des lignes numérotées de 1 à k donnent en fait les formes conjuguées des sous-tableaux formés des cases numérotées de 1 à k. Dans l'exemple précédent, ces sommes sont successivement $(1,1,1,0,0,0,0)$, $(2,2,1,1,0,0,0)$, $(3,2,1,1,0,0,0)$, $(3,2,1,1,1,1,0)$, $(3,3,1,1,1,1,0)$, $(4,3,2,1,1,1,0)$, $(4,3,2,1,1,1,1)$ et enfin $(5,3,2,1,1,1,1) = \lambda'$.

Enumérer les tableaux de Young de forme λ et poids μ fixés revient donc à énumérer les matrices de 0 et de 1 ayant $l(\mu)$ lignes et λ_1 colonnes, vérifiant cette propriété.

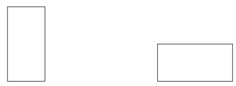
Les tableaux de Young ont donné naissance récemment [12] à des généralisations, les *tableaux de rubans*, dont l'apport en théorie des représentations se clarifie jour après jour.

7.1 Tableaux de rubans

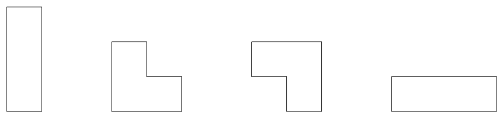
Les *tableaux de rubans*, introduits par Stanton et White [12] en 1985, peuvent être utilisés pour construire une base de vecteurs de plus haut poids dans l'espace de Fock de l'algèbre quantique affine $U_q(\widehat{sl}_n)$ [6]. Cette base joue un rôle crucial dans le calcul de la base canonique de la q-déformation de l'espace de Fock. Dans [9], Leclerc et Thibon ont conjecturé que cette base donnait les matrices de décomposition des q-algèbres de Schur aux racines de l'unité. Par ailleurs, les fonctions génératrices des tableaux de rubans conduisent à des généralisations des fonctions de Hall-Littlewood, qui sont des q-analogues des produits de fonctions de Schur [6].

Les tableaux de rubans d'une forme donnée et d'un poids donné peuvent être engendrés récursivement et l'algorithme que nous décrivons est bien adapté à une parallélisation immédiate. Nous avons implanté cet algorithme afin qu'il puisse s'exécuter au sein d'une machine virtuelle. Le programme calcule essentiellement la fonction génératrice des tableaux de rubans d'une forme et d'un poids donnés en fonction d'une statistique appelée *spin*. Nous avons ainsi obtenu un algorithme performant de génération de tableaux de rubans qui repousse les limites des polynômes de spin couramment calculés.

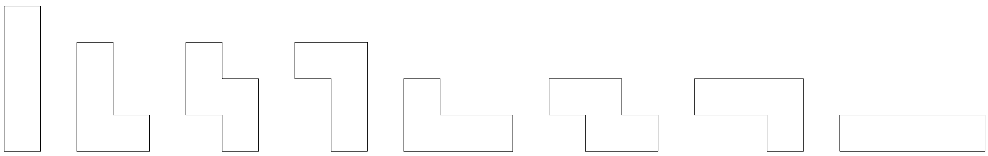
Nous nous intéressons donc désormais au remplissage d'une forme—un diagramme de Ferrers—avec des *rubans*. Un *k-ruban* est un diagramme connexe de k boîtes (ne contenant pas de carré 2×2 de boîtes) tel que l'on puisse aller d'une boîte à une autre par un chemin de boîtes ayant un côté en commun et tel que si l'on parcourt le ruban de gauche à droite, on ait le choix entre avancer horizontalement (vers la droite) ou verticalement vers le bas. Il est aisé de vérifier qu'il existe exactement 2^{k-1} k-rubans différents pour un k fixé. Par exemple, il y a 2 dominos correspondant à $k = 2$:



4 formes différentes de 3-rubans ($k = 3$) :



ou encore les 8 formes suivantes pour $k = 4$:



Tous les diagrammes de Ferrers ne sont pas pavables par des k-rubans. Il est clair qu'une condition évidente est que le nombre de boîtes du diagramme doit être divisible par k mais il est tout aussi clair que cette condition n'est pas suffisante. Par exemple, la forme $(3, 2, 1)$ n'est pas pavable par des dominos.

La *racine* d'un ruban est la boîte la plus en bas à droite du ruban comme le montre la figure 7.1 où la racine est marquée d'un point noir :

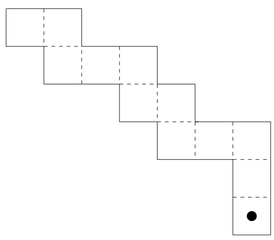


Figure 7.1: *Un exemple de 12-ruban avec sa racine marquée d'un point noir.*

Partant du pavage d'une forme donnée λ avec des k-rubans, on associe à chacun des rubans un entier positif comme dans le cas des boîtes des tableaux de Young, c'est-à-dire de telle sorte que les lectures ligne de ces entiers forment des suites croissantes au sens large et que les lectures colonnes donnent des suites strictement croissantes.

La lecture des colonnes mérite d'être expliquée. La condition exacte de lecture des colonnes du tableau de rubans spécifie en fait qu'il ne doit pas exister une quelconque

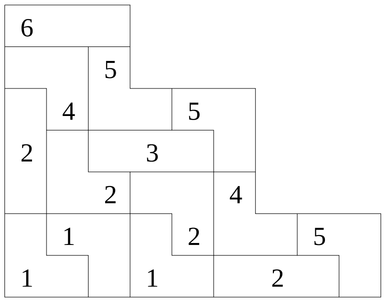
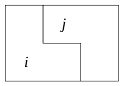
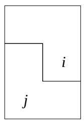


Figure 7.2: *Un exemple de tableau de* 3*-rubans.*

boîte d'un ruban numéroté j au dessous d'une racine de ruban numérotée $i \leq j$. Par exemple, les configurations suivantes de deux 3-rubans numérotés respectivement i et j sont autorisées pour $i \leq j$:



alors que les agencements correspondant à la figure suivante sont interdits :



puisque la racine d'un ruban i est au dessus d'une boîte d'un ruban numéroté $j \geq i$.

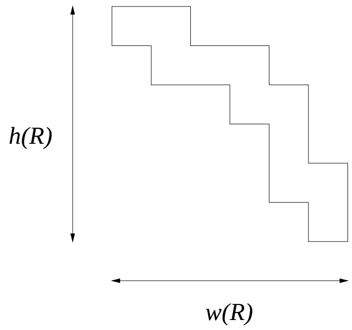
Remarque. On note généralement l'étiquette d'un k-ruban dans une seule des boîtes de ce ruban. La position de l'étiquette dépend en plus de la position du ruban au sein du tableau de rubans qui le contient. Nous verrons bientôt quelle est la convention adoptée pour le positionnement des numéros de rubans, convention qui facilite d'ailleurs certains codages (décodages) des tableaux de rubans.

Le *poids* d'un tableau de rubans est défini comme dans le cas d'un tableau de Young. Par exemple, la figure 7.2 montre un tableau de 3-rubans, de forme $\lambda = (9, 9, 6, 6, 6, 3, 3)$ et de poids $\mu = (3, 4, 1, 2, 3, 1)$.

7.2 Polynôme de spin

Alors qu'une des statistiques fondamentales dans le cas des tableaux de Young est la charge (ou de manière équivalente, la cocharge), la statistique maîtresse pour les tableaux de rubans est appelée *spin*.

Soit R un k-ruban, on note par $h(R)$ sa hauteur et par $w(R)$ sa largeur comme le montre la figure suivante :



A noter au passage la propriété élémentaire :

$$h(R) + w(R) = k + 1 \quad . \tag{7.3}$$

Le *spin*[2] $s(R)$ du ruban R est alors défini par :

$$s(R) \;:=\; \frac{h(R) - 1}{2} \quad . \tag{7.4}$$

Le spin d'un tableau de rubans T est alors défini comme la somme des spins des différents rubans qui le composent :

$$s(T) \;:=\; \sum_{R \in T} s(R) \quad . \tag{7.5}$$

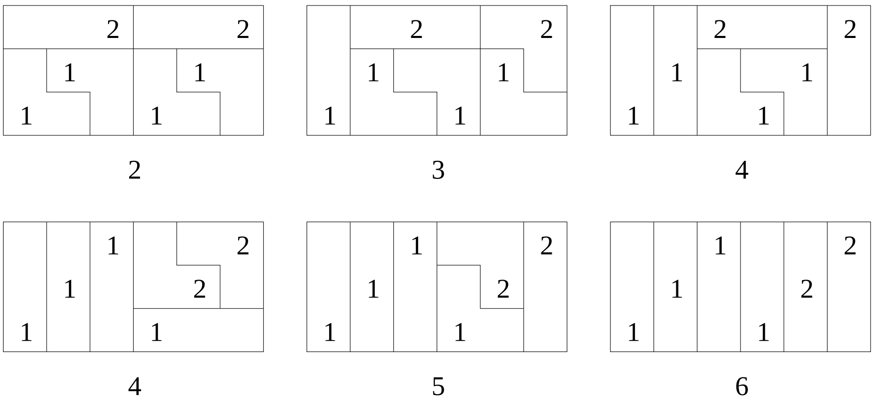
Par exemple, le ruban de la figure 7.1 a un spin de 5/2 et le tableau de rubans de la figure 7.2 a un spin égal à 6.

Soient λ et μ deux partitions telles que $|\lambda| = k|\mu|$. On considère alors l'ensemble $\mathrm{Tab}_k(\lambda, \mu)$ de tous les tableaux de k-rubans de forme λ et de poids μ. L'information la plus importante qui concerne cet ensemble de tableaux est le *polynôme de spin* $G^{(k)}_{\lambda,\mu}(q)$ défini par :

$$G^{(k)}_{\lambda,\mu}(q) \;:=\; \sum_{T \in \mathrm{Tab}_k(\lambda,\mu)} q^{s(T)} \quad . \tag{7.6}$$

En d'autres termes, ce polynôme est la fonction génératrice du nombre de tableaux de rubans d'une forme et d'un poids donnés en fonction du spin. L'exemple suivant illustre la définition de ce polynôme.

Exemple 7.1 Soient la forme $\lambda = (6, 6, 6)$ et le poids $\mu = (4, 2)$. Il existe 6 tableaux de rubans de forme λ et de poids μ composés de 3-rubans, comme le montre la figure suivante :



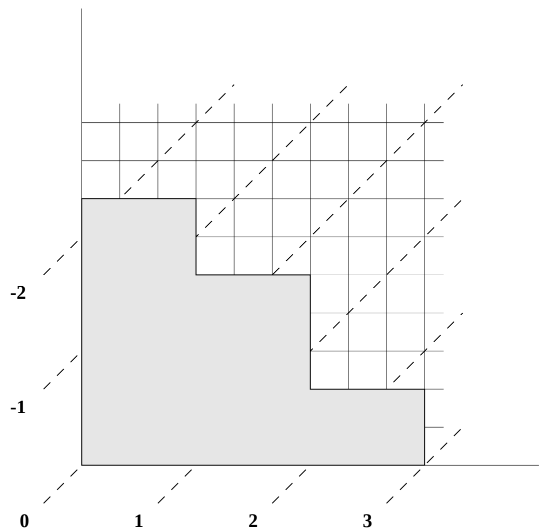
[2]La statistique spin est à valeur dans $\frac{1}{2}\mathbb{N}$.

dans laquelle nous avons indiqué en dessous de chaque tableau le spin correspondant. Le polynôme de spin $G^{(3)}_{(6,6,6),(4,2)}(q)$ associé à cet ensemble est donc :

$$G^{(3)}_{(6,6,6),(4,2)}(q) = q^2 + q^3 + 2q^4 + q^5 + q^6 \quad ,$$

puisque par exemple il y a deux tableaux de spin 4.

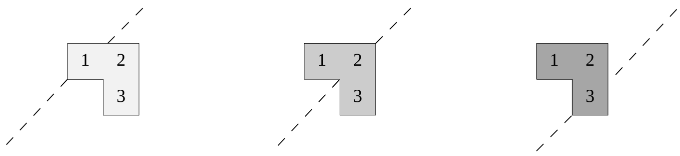
L'exemple précédent nous amène à faire le point sur l'emplacement de la numérotation des rubans. Etant donné un k-ruban, nous avons k positions possibles pour inscrire son numéro puisque le ruban est constitué de k boîtes. Cependant, les 6 tableaux précédents montrent que les positions des numéros ne sont pas aléatoires mais qu'au contraire la tendance est à une numérotation le long de diagonales. En fait, on numérote les diagonales d'un tableau de k-rubans comme le montre la figure suivante pour $k = 3$:



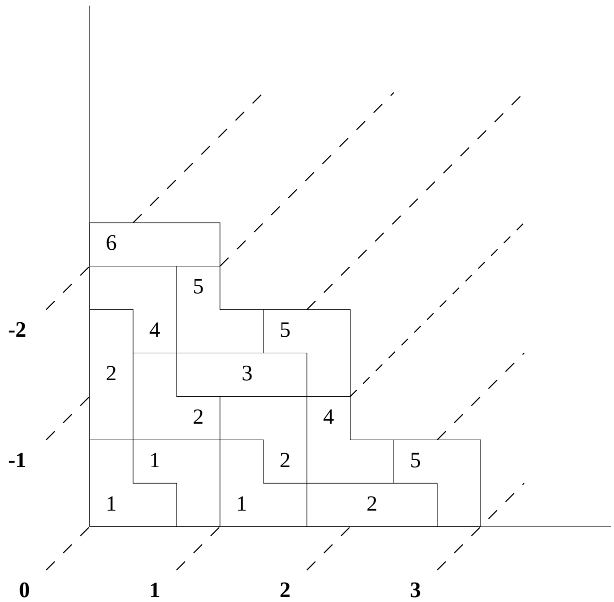
Maintenant, il est clair que chaque ruban n'est coupé que par une seule diagonale et c'est justement dans la boîte du ruban traversée par la diagonale que l'on inscrit le numéro du ruban.

Remarque. La numérotation des diagonales d'un tableau de Young peut être vue comme la numérotation des diagonales d'un tableau de 1-rubans (rubans constitués d'une seule boîte).

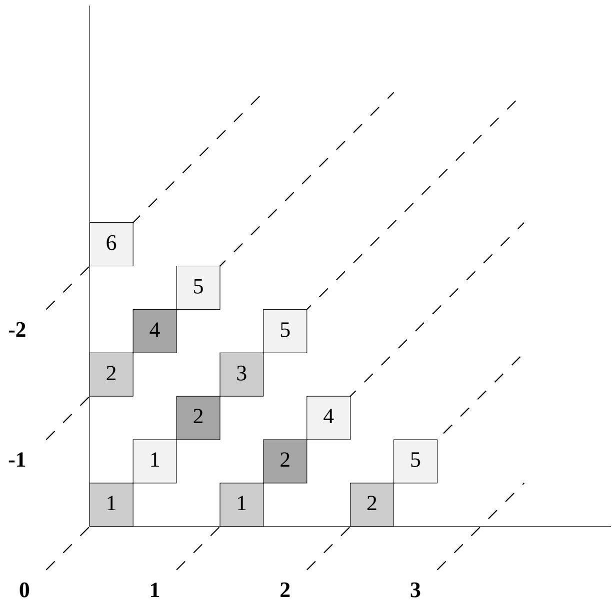
Avant d'aborder le codage matriciel des tableaux de rubans, nous présentons un codage original dû à Stanton et White mettant en évidence l'importance de ces diagonales. Ce codage nous dit que tout tableau de k-rubans peut être codé par k tableaux de Young. Pour cela, il faut associer à chaque ruban du tableau de k-rubans une couleur numérotée de 1 à k. Pour être simple, on peut colorier les boîtes d'un k-ruban de gauche à droite et de bas en haut puis dire que la couleur globale du ruban est la couleur de la boîte traversée par l'unique diagonale qui coupe le ruban. La figure suivante montre les trois configurations d'intersection d'un 3-ruban avec une diagonale conduisant aux trois couleurs de rubans possibles :



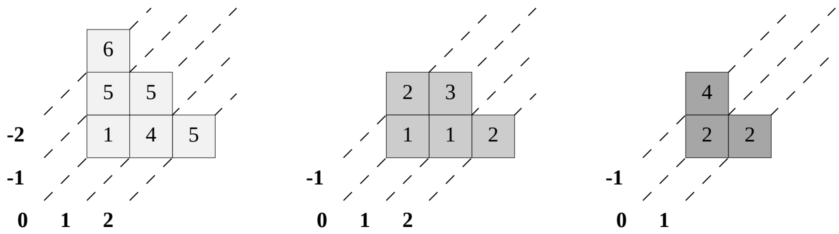
Chacun des k tableaux de Young associés au tableau de k-rubans va alors correspondre à une des k couleurs et chaque ruban contribuera à la construction du tableau de Young correspondant à sa couleur. La construction repose uniquement sur une lecture des diagonales du tableau de rubans. Si nous reprenons l'exemple de la figure 7.2 :



nous ne gardons que les boîtes touchées par une diagonale et associons à ces boîtes une couleur comme expliqué précédemment, nous obtenons :



La diagonale de numéro d du tableau de Young correspondant à la couleur c est obtenue par lecture de la diagonale de même numéro du tableau de rubans en ne tenant compte que des rubans de couleur c. Cet algorithme conduit aux trois tableaux de Young suivants dans le cas du tableau de rubans de la figure 7.2 :



Un tableau de k-rubans peut donc être codé par k matrices de 0 et de 1 mais nous allons voir un codage plus proche du codage matriciel direct des tableaux de Young.

7.3 Codage matriciel des tableaux de rubans

On peut coder un tableau de rubans T de forme λ et de poids μ par une matrice $M(T)$ ayant r lignes et s colonnes, r étant la longueur de la partition λ et s sa plus grande part (ou de manière similaire, la longueur de la partition conjuguée de λ). La matrice est remplie avec les valeurs 0 ou k, la i-ième ligne contenant la valeur k exactement μ_i fois (comme dans le cas du codage des tableaux de Young). Pour obtenir la i-ième ligne de $M(T)$ (en numérotant les lignes à partir du bas) on regarde les indices de colonnes contenant la racine d'un ruban numéroté i et on inscrit k dans la colonne correspondante de la matrice. Les autres entrées sont positionnées à 0. La matrice codant le tableau de 3-rubans de la figure 7.2 est la suivante :

$$\begin{array}{ccccccccc}
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 & 3 & 0 & 0 & 3 \\
0 & 3 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\
3 & 0 & 3 & 0 & 3 & 0 & 0 & 3 & 0 \\
0 & 3 & 3 & 0 & 3 & 0 & 0 & 0 & 0
\end{array}$$

Par exemple, la figure 7.2 précise que les racines des rubans numérotés 2 se trouvent dans les colonnes 1, 3, 5 et 8 de sorte que la ligne correspondant aux rubans numérotés 2 est :

$$3 \quad 0 \quad 3 \quad 0 \quad 3 \quad 0 \quad 0 \quad 3 \quad 0$$

✍**Remarque**. Bien entendu, comme dans le cas des tableaux de Young, la lecture du nombre de k par ligne donne le poids du tableau de rubans.

Il faut noter que l'utilisation de la lettre k dans la matrice n'apporte pas spécialement d'information si ce n'est qu'elle rappelle que la matrice code un tableau composé de k-rubans. En fait, on pourrait tout aussi bien coder un tel tableau par une matrice à valeur binaire et stocker séparément la valeur k.

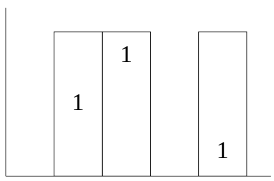
Exemple 7.2 Dans cet exemple, nous reprenons les 6 tableaux de l'exemple 7.1 et donnons les matrices correspondant à chacun d'eux :

$$\begin{matrix} 0 & 0 & 3 & 0 & 0 & 3 \\ 0 & 3 & 3 & 0 & 3 & 3 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 3 & 0 & 3 \\ 3 & 0 & 3 & 3 & 0 & 3 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 3 & 3 \\ 3 & 3 & 0 & 3 & 3 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 & 3 & 3 \\ 3 & 3 & 3 & 0 & 0 & 3 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 3 & 3 \\ 3 & 3 & 3 & 0 & 3 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 & 0 & 3 & 3 \\ 3 & 3 & 3 & 3 & 0 & 0 \end{matrix}$$

7.4 Algorithme de décodage

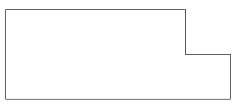
Pour reconstruire le tableau de ruban T à partir de sa matrice $M(T)$, on commence par les rubans numérotés 1. La dernière ligne de la matrice précise les emplacements des racines de ces rubans. Par exemple, pour le cas du tableau de rubans de la figure 7.2 on obtient la configuration de rubans suivante :



mais nous devons "replier" les rubans. Pour ce faire, nous procédons comme suit : on considère le vecteur $\rho_s := [s-1, s-2, \ldots, 2, 1, 0]$ où s est le nombre de colonnes de la matrice et nous additionnons ce vecteur à la dernière ligne de la matrice $M(T)$. Cela donne pour notre exemple :

$$\begin{array}{ccccccccccc} & 0 & 3 & 3 & 0 & 3 & 0 & 0 & 0 & 0 \\ + & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline = & 8 & 10 & 9 & 5 & 7 & 3 & 2 & 1 & 0 \end{array}$$

Puis nous trions le vecteur obtenu en ordre décroissant et soustrayons ρ_s. Cette opération produit la forme conjuguée de la forme occupée par les rubans numérotés 1. Notre exemple conduit au vecteur trié $[10, 9, 8, 7, 5, 3, 2, 1, 0]$ puis en soustrayant ρ_9, on obtient la partition $[2, 2, 2, 2, 1, 0, 0, 0, 0]$:



2 2 2 2 1

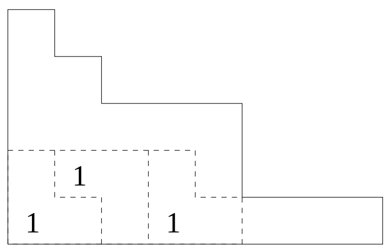
qui est bien le conjugué de la forme occupée par les rubans 1 de la figure 7.2.

Remarque. On peut vérifier qu'il existe au plus un pavage d'une forme donnée par des rubans de mêmes numéros.

Pour reconstruire la forme occupée par les rubans numérotés 2, on part du vecteur trié obtenu à l'étape précédente et on lui ajoute la ligne de la matrice $M(T)$ correspondant à ces rubans (avant dernière ligne). Ensuite, on procède comme expliqué dans le cas des rubans numérotés 1 afin d'obtenir la forme (conjuguée) occupée par les rubans numérotés 1 et 2 mais puisque l'on connaît la forme (conjuguée) occupée par les rubans 1 on déduit sans peine celle occupée par les rubans 2. Pour notre exemple, ce calcul donne :

$$\begin{array}{cccccccccc} & 3 & 0 & 3 & 0 & 3 & 0 & 0 & 3 & 0 \\ + & 10 & 9 & 8 & 7 & 5 & 3 & 2 & 1 & 0 \\ \hline = & 13 & 9 & 11 & 7 & 8 & 3 & 2 & 4 & 0 \end{array}$$

vecteur que nous réordonnons en $[13, 11, 9, 8, 7, 4, 3, 2, 0]$. Par soustraction de ρ_9, on trouve la partition $\nu' = [5, 4, 3, 3, 3, 1, 1, 1, 0]$ qui est bien à nouveau la partition conjuguée de la forme occupée par les rubans numérotés 1 et 2 dans le tableau de rubans de la figure 7.2 :



5 4 3 3 3 1 1 1

On répète cette construction pour les rubans numérotés 3, 4 et ainsi de suite.

7.5 Calcul récursif du polynôme de spin

Un avantage clair de ce décodage est qu'il permet de calculer à chaque étape le spin du tableau de ruban correspondant. Nous rappelons qu'une des étapes du décodage consiste à ajouter le vecteur ρ_s au vecteur obtenu à l'étape précédente et à réordonner ce vecteur. En fait, c'est exactement à ce niveau que l'on peut extraire l'information concernant le spin des rubans que l'on est en train de replier et *a fortiori* déduire le spin global du tableau de rubans.

L'idée est de noter à chaque étape le nombre d'inversions I du vecteur obtenu comme somme du vecteur de l'étape précédente et de ρ_s. Ce nombre d'inversions est le nombre d'étapes élémentaires d'un tri à bulle qui réordonnerait le vecteur en ordre décroissant. Notre exemple conduit au vecteur $[8, 10, 9, 5, 7, 3, 2, 1, 0]$ que l'on réordonne grâce à $I = 3$ étapes élémentaires qui sont par exemple les suivantes :

$$\begin{array}{ccccccccc}
8 & 10 & 9 & 5 & 7 & 3 & 2 & 1 & 0 \\
10 & 8 & 9 & 5 & 7 & 3 & 2 & 1 & 0 \\
10 & 9 & 8 & 5 & 7 & 3 & 2 & 1 & 0 \\
10 & 9 & 8 & 7 & 5 & 3 & 2 & 1 & 0
\end{array}$$

Le nombre d'inversions est en fait égal à :

$$I = \sum_R (w(R) - 1) \quad , \tag{7.7}$$

puisqu'à chaque étape la largeur d'un ruban est augmentée de 1 et qu'au départ tous les rubans sont verticaux c'est-à-dire de largeur 1. Enfin, grâce à (7.4) et (7.7), nous obtenons une expression de I en fonction de k et des hauteurs de rubans :

$$I = \sum_R (w(R) - 1) = \sum_R (k - h(R)) \quad ,$$

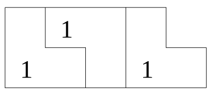
formule qui conduit immédiatement à l'expression du spin $s(T)$ du tableau de rubans T en fonction de I. Soit n le nombre de k-rubans dans le tableau de rubans T, alors puisque $h(R) = 2s(R) + 1$:

$$I = \sum_R (k - (2s(R) + 1)) = nk - n - 2\sum_R s(R) \quad ,$$

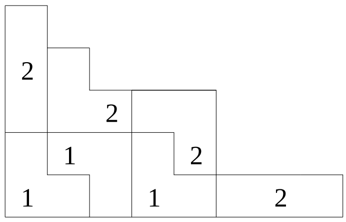
d'où l'on extrait $s(T) = \sum_R s(R)$:

$$s(T) = \frac{n(k-1) - I}{2} \quad . \tag{7.8}$$

Pour illustrer ce calcul, nous l'appliquons à notre exemple dans lequel tous les rubans numérotés 2 ont une largeur de 2 de sorte que $I = 3 = (2-1) + (2-1) + (2-1)$ et le spin du tableau de rubans composé uniquement des rubans numérotés 1 :



est donc $s(T) = \frac{3}{2}$. Dans notre algorithme, l'étape suivante décrit l'ajout des rubans numérotés 2 et conduit au vecteur $[13, 9, 11, 7, 8, 3, 2, 4, 0]$ réordonné en ordre décroissant en $[13, 11, 9, 8, 7, 4, 3, 2, 0]$ à l'aide de 4 transpositions élémentaires. Maintenant $I = 3 + 4$ car I est toujours égal à la somme $\sum_R (w(R) - 1)$ mais cette somme est prise sur tous les rubans numérotés 1 et 2. Le spin du tableau partiel :



est donc égal à $s(T) = \frac{7(3-1)-I}{2} = \frac{7}{2}$.

Nous disposons maintenant de tous les éléments qui vont nous permettre de calculer récursivement l'ensemble $\mathrm{Tab}_k(\lambda,\mu)$ de tous les tableaux de k-rubans de forme donnée λ et de poids fixé μ en même temps que le polynôme de spin correspondant. En effet, le codage précédent des tableaux de rubans par des matrices conduit à une formule récursive de calcul des polynômes suivants à partir desquels nous allons déduire les polynômes de spin $G^{(k)}_{\lambda,\mu}(q)$.

Dans la formule suivante, α dénote la somme $\lambda' + \rho_s$ avec $s \geq \lambda_1$:

$$F^{(k)}_{\alpha,\mu}(q) := \sum_{T\in \mathrm{Tab}_k(\lambda,\mu)} q^I = \sum_{T\in \mathrm{Tab}_k(\lambda,\mu)} \Big(\prod_{R\in T} q^{(w(R)-1)} \Big) \quad , \tag{7.9}$$

où $I = \sum_{R\in T} (w(R)-1)$.

Soit $\mu = (\mu_1,\ldots,\mu_r)$. On obtient les positions des μ_r rubans numérotés r dans le tableau de rubans de forme λ et de poids μ en renversant l'algorithme de décodage décrit précédemment. L'ensemble des formes globales possibles contenant les rubans numérotés de 1 à $r-1$ est obtenu en soustrayant à α toutes les permutations distinctes du vecteur :

$$v_r := [0,\ldots,0,k,\ldots,k] \quad ,$$

contenant μ_r composantes égales à k. On trie alors ces vecteurs en ordre décroissant en gardant les différents nombres d'inversions qui nous permettront de calculer le spin des tableaux obtenus. On construit ainsi un arbre de racine α, dont les branches sont indicées par les permutations distinctes v_r^τ de v_r et dont chaque sommet est le résultat du tri d'un vecteur $\alpha - v_r^\tau$ accompagné du nombre d'inversions correspondant à la permutation triante.

Remarque. Si un sommet contient un vecteur ayant des répétitions alors on peut montrer que ce vecteur ne peut appartenir à la matrice décrivant un tableau de rubans, de sorte que toutes les branches menant à un tel type de vecteur ayant des répétitions de valeurs sont trivialement éliminées.

Soit $\alpha_1,\ldots,\alpha_m$ les vecteurs restants (au sens de non éliminés trivialement) et $I_1,\ldots,I_m$ les nombres d'inversions correspondants. On note par μ^* le vecteur obtenu à partir de μ en supprimant la dernière composante :

$$\mu^* := (\mu_1,\ldots,\mu_{r-1}) \quad .$$

La formule de récurrence est alors la suivante :

$$F^{(k)}_{\alpha,\mu}(q) := \sum_{p=1}^{m} q^{I_p}\, F^{(k)}_{\alpha_p,\mu^*}(q) \quad , \tag{7.10}$$

avec la condition supplémentaire que $F^{(k)}_{\rho_s,\emptyset} := 1$. Le polynôme de spin est finalement donné par :

$$
\begin{aligned}
G^{(k)}_{\lambda,\mu}(q) &= \sum_{T\in \mathrm{Tab}_k(\lambda,\mu)} q^{s(T)} \\
&= \sum_{T\in \mathrm{Tab}_k(\lambda,\mu)} q^{\frac{|\mu|(k-1)-I}{2}} \\
&= q^{\frac{|\mu|(k-1)}{2}} \sum_{T\in \mathrm{Tab}_k(\lambda,\mu)} q^{-\frac{I}{2}} \quad ,
\end{aligned}
$$

c'est-à-dire :

$$
G^{(k)}_{\lambda,\mu}(q) = q^{\frac{|\mu|(k-1)}{2}} F^{(k)}_{\alpha,\mu}(q^{-\frac{1}{2}}) \quad . \tag{7.11}
$$

Remarque. Cette formule produit réellement un polynôme (sans exposant fractionnaire) lorsque les parts de λ sont divisibles par k. Par ailleurs, il semble que ce cas soit le plus intéressant.

Exemple 7.3 Nous reprenons encore l'exemple 7.1 et montrons le déroulement complet de l'algorithme d'énumération des 6 tableaux. Rappelons que nous souhaitons énumérer les tableaux de 3-rubans de forme $\lambda = (6,6,6)$ et de poids $\mu = (4,2)$. On a donc $\lambda' = (3,3,3,3,3,3)$, $\rho_6 = [5,4,3,2,1,0]$ et finalement $\alpha = [3+5, 3+4, 3+3, 3+2, 3+1, 3+0]$.

On calcule alors les $\binom{6}{2} = 15$ permutations distinctes $v_2\tau$ du vecteur $v_2 = [0,0,0,0,3,3]$ et l'on regarde les vecteurs $\alpha - v_2\tau$. Seuls les 6 vecteurs suivants ne sont pas trivialement éliminés—car ne contenant pas de répétition de valeurs ou de valeur négative :

$$
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 0 & 0 & 0 & 0 & 3 & 3 \\
\hline
= & 8 & 7 & 6 & 5 & 1 & 0
\end{array}
\qquad
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 0 & 0 & 0 & 3 & 0 & 3 \\
\hline
= & 8 & 7 & 6 & 2 & 4 & 0
\end{array}
\qquad
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 0 & 0 & 3 & 0 & 0 & 3 \\
\hline
= & 8 & 7 & 3 & 5 & 4 & 0
\end{array}
$$

$$
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 0 & 0 & 0 & 3 & 3 & 0 \\
\hline
= & 8 & 7 & 6 & 2 & 1 & 3
\end{array}
\qquad
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 0 & 3 & 0 & 0 & 3 & 0 \\
\hline
= & 8 & 4 & 6 & 5 & 1 & 3
\end{array}
\qquad
\begin{array}{rcccccc}
 & 8 & 7 & 6 & 5 & 4 & 3 \\
- & 3 & 0 & 0 & 3 & 0 & 0 \\
\hline
= & 5 & 7 & 6 & 2 & 4 & 3
\end{array}
$$

On trie maintenant ces vecteurs en ordre décroissant en stockant les nombres d'inversions correspondants $I_1, \ldots, I_6$ indiqués en gras à gauche des vecteurs réordonnés :

$$
\begin{array}{lllll}
\mathbf{0}, & 8\ 7\ 6\ 5\ 1\ 0 & \mathbf{1}, & 8\ 7\ 6\ 4\ 2\ 0 & \mathbf{2}, \ 8\ 7\ 5\ 4\ 3\ 0 \\
\mathbf{3}, & 8\ 7\ 6\ 3\ 2\ 1 & \mathbf{3}, & 8\ 6\ 5\ 4\ 3\ 1 & \mathbf{4}, \ 7\ 6\ 5\ 4\ 3\ 2
\end{array}
$$

On calcule ensuite les $\binom{6}{4} = 15$ permutations distinctes $v_1\tau$ de $v_1 = [0,0,3,3,3,3]$ et partant de chacun des vecteurs $n_1, \ldots, n_6$ des 6 nœuds de l'arbre d'énumération, on cherche les vecteurs $n_i - v_1\tau$ qui sont des permutations de ρ_6. Il n'est pas difficile de se persuader que chaque nœud de l'avant dernier niveau de l'arbre (c'est-à-dire juste avant le cas de base $F^{(3)}_{\rho_6,\emptyset}$) ne peut donner naissance qu'à au plus un successeur, lequel doit par essence être une permutation du vecteur ρ_6. En d'autres termes, l'on sait dès à présent qu'il y aura au plus 6 solutions à notre problème. En fait, pour chaque n_i un seul vecteur $v_1\tau$ parmi les 15 permet de produire un $n_i - v_1\tau$ admissible et par conséquent il y a exactement 6 solutions comme le montre la figure suivante :

$$\begin{array}{rcccccc} & 8 & 7 & 6 & 5 & 1 & 0 \\ - & 3 & 3 & 3 & 3 & 0 & 0 \\ \hline = & 5 & 4 & 3 & 2 & 1 & 0 \end{array} \qquad \begin{array}{rcccccc} & 8 & 7 & 6 & 4 & 2 & 0 \\ - & 3 & 3 & 3 & 3 & 0 & 0 \\ \hline = & 5 & 4 & 3 & 1 & 2 & 0 \end{array} \qquad \begin{array}{rcccccc} & 8 & 7 & 5 & 4 & 3 & 0 \\ - & 3 & 3 & 3 & 3 & 0 & 0 \\ \hline = & 5 & 4 & 2 & 1 & 3 & 0 \end{array}$$

$$\begin{array}{rcccccc} & 8 & 7 & 6 & 3 & 2 & 1 \\ - & 3 & 3 & 3 & 3 & 0 & 0 \\ \hline = & 5 & 4 & 3 & 0 & 2 & 1 \end{array} \qquad \begin{array}{rcccccc} & 8 & 6 & 5 & 4 & 3 & 1 \\ - & 3 & 3 & 3 & 0 & 3 & 0 \\ \hline = & 5 & 3 & 2 & 4 & 0 & 1 \end{array} \qquad \begin{array}{rcccccc} & 7 & 6 & 5 & 4 & 3 & 2 \\ - & 3 & 3 & 0 & 3 & 3 & 0 \\ \hline = & 4 & 3 & 5 & 1 & 0 & 2 \end{array}$$

On trie à nouveau ces vecteurs en ordre décroissant en additionnant les nouveaux nombres d'inversions avec les nombres correspondants $I_1, \ldots, I_6$ de l'étape précédente :

$$\begin{array}{lllll} \mathbf{0}, & 5\ 4\ 3\ 2\ 1\ 0 & \mathbf{2}, & 5\ 4\ 3\ 2\ 1\ 0 & \mathbf{4}, \ 5\ 4\ 3\ 2\ 1\ 0 \\ \mathbf{4}, & 5\ 4\ 3\ 2\ 1\ 0 & \mathbf{6}, & 5\ 4\ 3\ 2\ 1\ 0 & \mathbf{8}, \ 5\ 4\ 3\ 2\ 1\ 0 \end{array}$$

Nous résumons ces résultats en présentant les matrices ainsi engendrées, accompagnées du nombre total d'inversions associé aux différents tableaux de rubans :

$$\begin{array}{ccccccc} \mathbf{0} & 0 & 0 & 0 & 0 & 3 & 3 \\ & 3 & 3 & 3 & 3 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} \mathbf{2} & 0 & 0 & 0 & 3 & 0 & 3 \\ & 3 & 3 & 3 & 3 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} \mathbf{4} & 0 & 0 & 3 & 0 & 0 & 3 \\ & 3 & 3 & 3 & 3 & 0 & 0 \end{array}$$

$$\begin{array}{ccccccc} \mathbf{4} & 0 & 0 & 0 & 3 & 3 & 0 \\ & 3 & 3 & 3 & 3 & 0 & 0 \end{array} \qquad \begin{array}{ccccccc} \mathbf{6} & 0 & 3 & 0 & 0 & 3 & 0 \\ & 3 & 3 & 3 & 0 & 3 & 0 \end{array} \qquad \begin{array}{ccccccc} \mathbf{8} & 3 & 0 & 0 & 3 & 0 & 0 \\ & 3 & 3 & 0 & 3 & 3 & 0 \end{array}$$

On déduit immédiatement que :

$$F^{(3)}_{(8,7,6,5,4,3),(4,2)}(q) = q^0 + q^2 + 2q^4 + q^6 + q^8 \quad ,$$

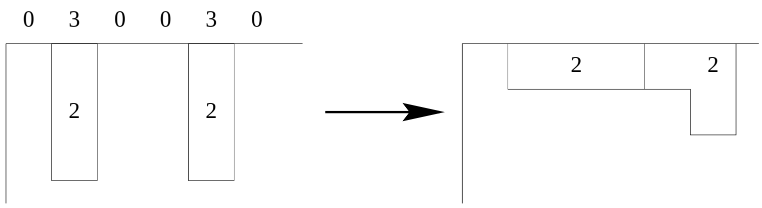
et finalement que le polynôme de spin associé à cet ensemble de 6 tableaux de rubans est :

$$\begin{aligned} G^{(3)}_{(6,6,6),(4,2)}(q) &= q^{\frac{6(3-1)}{2}}(1 + q^{-\frac{2}{2}} + 2q^{-\frac{4}{2}} + q^{-\frac{6}{2}} + q^{-\frac{8}{2}}) \\ &= q^6 + q^5 + 2q^4 + q^3 + q^2 \quad . \end{aligned}$$

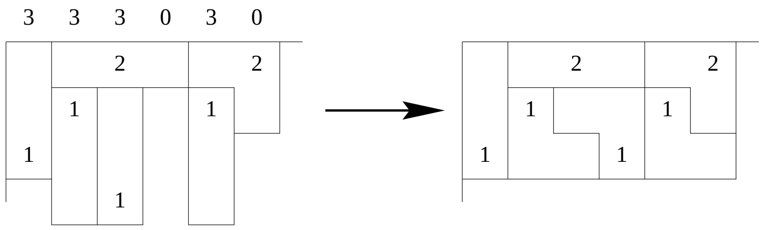
Les 6 matrices ci-dessus ne sont pas égales aux matrices données dans l'exemple 7.2. En fait, nous avons précisé précédemment que pour énumérer des tableaux de rubans nous devions renverser l'algorithme de décodage. Nous avons à ce sujet omis de mentionner que ce renversement avait en quelque sorte des effets secondaires ! Si l'on appelle *sommet* d'un ruban R la boîte la plus en haut à gauche de R, il faut considérer que les matrices produites par l'algorithme d'énumération décrivent les emplacements des sommets et non plus des racines des rubans impliqués. Par ailleurs, l'opération de repliage (correspondant au tri d'un vecteur) est modifiée en ce sens qu'il faut voir les rubans comme pendus par leur sommet, et chercher à les replier vers le haut à droite cette fois. Prenons l'exemple de la matrice :

$$\begin{array}{cccccc} 0 & 3 & 0 & 0 & 3 & 0 \\ 3 & 3 & 3 & 0 & 3 & 0 \end{array}$$

Les rubans numérotés 2 sont suspendus par leur sommet puis repliés comme l'indique la figure suivante :



Puis c'est au tour des rubans numérotés 1 :



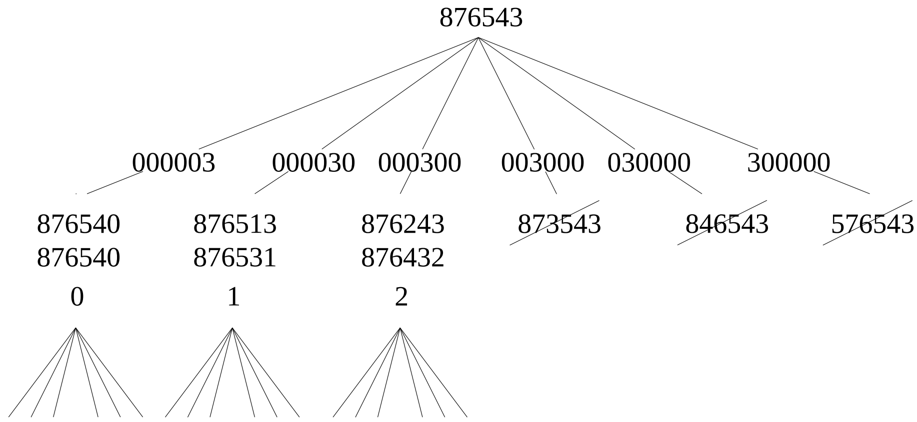
procédé qui conduit bien à l'obtention du tableau de rubans de spin 3 de l'exemple 7.1.

Voyons un autre exemple très proche de l'exemple précédent mais faisant apparaître une propriété intéressante de l'ensemble des nœuds construits au cours de l'énumération.

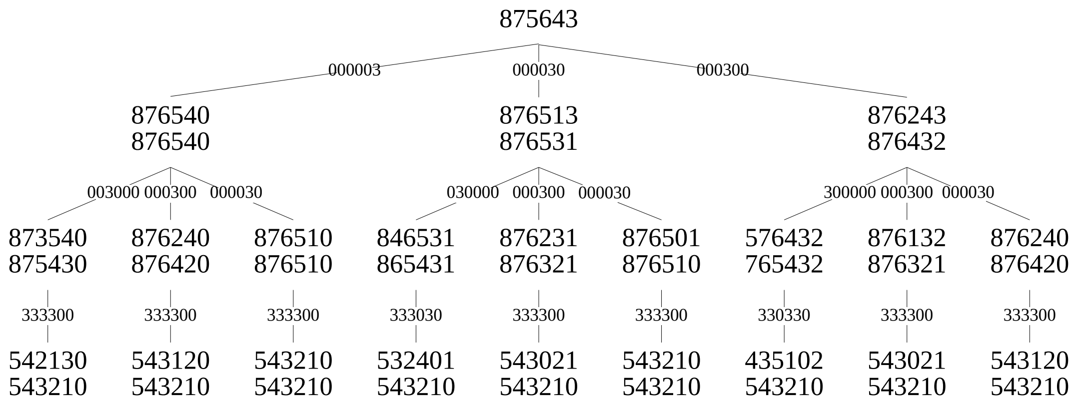
Exemple 7.4 Pour énumérer tous les tableaux de 3-rubans de forme $\lambda = (6,6,6)$ et de poids $\mu = (4,1,1)$ cette fois, nous calculons à nouveau $\lambda' = (3,3,3,3,3,3)$ et posons $s = 6$ *i.e.* le nombre de colonnes des matrices. Le vecteur α est toujours égal à $\alpha = [3,3,3,3,3,3] + [5,4,3,2,1,0] = [8,7,6,5,4,3]$ et correspond à la racine de l'arbre. Puisque le nombre de parts de μ est $r = 3$, le vecteur v_3 est initialisé à $[0,0,0,0,0,3]$, vecteur à partir duquel on engendre 6 permutations distinctes conduisant en fait à seulement 3 possibilités admissibles. Ainsi, nous savons déjà que :

$$F^{(3)}_{(8,7,6,5,4,3),(4,1,1)}(q) = F^{(3)}_{(8,7,6,5,4,0),(4,1)}(q) + q\,F^{(3)}_{(8,7,6,5,3,1),(4,1)}(q) + q^2\,F^{(3)}_{(8,7,6,4,3,2),(4,1)}(q) \quad .$$

La figure qui suit explicite les deux premiers niveaux de l'arbre d'énumération en faisant apparaître notamment les 3 vecteurs éliminés car contenant au moins une répétition de valeurs :

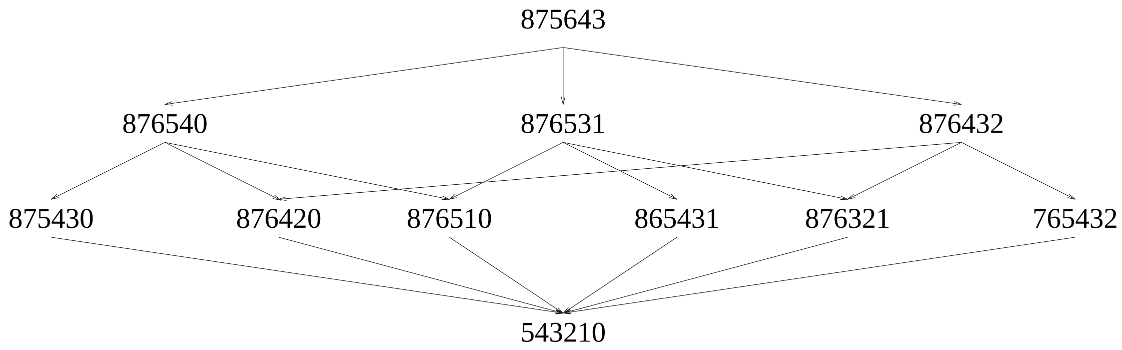


La figure suivante montre l'arbre complet d'énumération des tableaux de 3-rubans de forme $(6, 6, 6)$ et de poids $(4, 1, 1)$ en ne présentant que les branches conduisant effectivement à une solution :



Cette figure nous précise qu'il y a exactement 9 tableaux composés de 3-rubans, de forme $(6, 6, 6)$ et de poids $(4, 1, 1)$. Nous avons volontairement omis l'information concernant le nombre d'inversions associé à chaque nœud (cette information est somme toute très simple à retrouver à partir des informations contenues dans l'arbre) pour ne pas surcharger la figure mais plutôt mettre l'accent sur la propriété suivante.

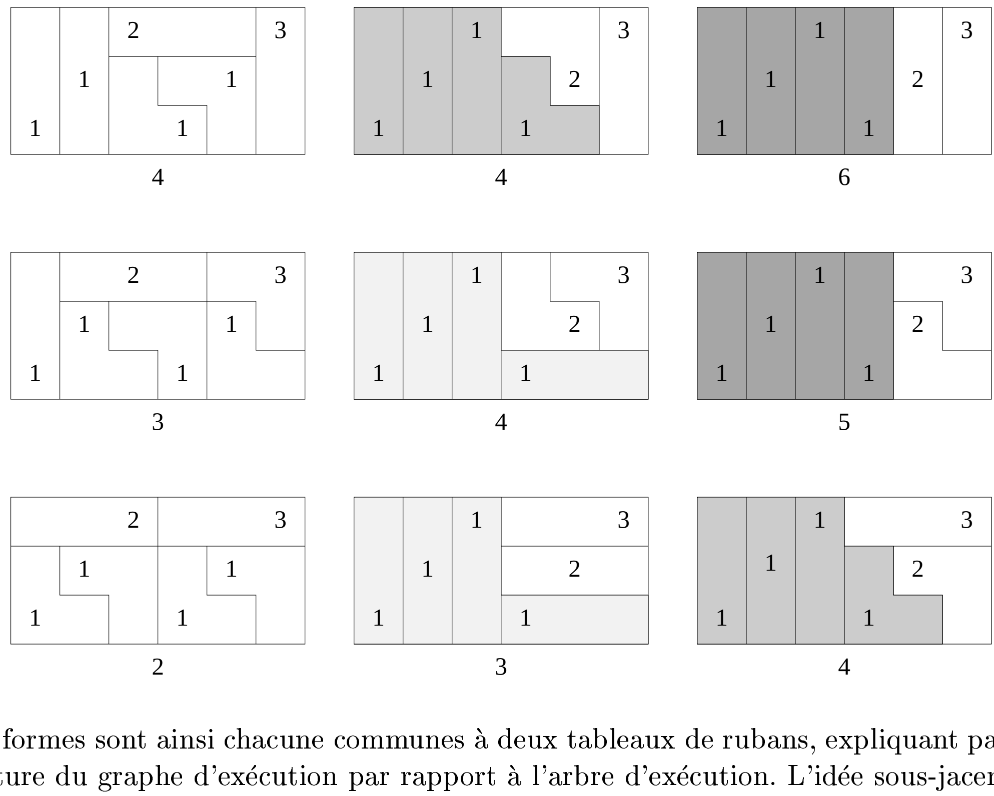
La remarque essentielle est que plusieurs branches conduisent à des nœuds identiques dans l'arbre, deux nœuds étant considérés identiques lorsque les deux vecteurs triés associés à ces deux nœuds sont égaux terme à terme. L'exemple le plus frappant est bien entendu que toutes les branches se terminent par le vecteur ρ_s mais ce n'est pas ce cas qui nous intéresse. En fait, c'est au cœur de l'arbre que tout se passe et qu'il peut arriver que plusieurs nœuds d'un même niveau de l'arbre soient les mêmes et par suite que les sous-arbres associés à ces nœuds soient exactement semblables. En d'autres termes, l'arbre d'énumération doit préférablement être vu comme un graphe orienté. Cela pour dire que le calcul effectif d'un ensemble de tableaux de rubans devra tenir compte de cette propriété et notamment l'exemple qui nous intéresse conduira à un graphe d'exécution qui ressemblera plutôt à la figure suivante :



Nous n'avons pas indiqué les différents nombres d'inversions pour chacun des nœuds des chemins menant de α à ρ_6 dans le graphe précédent, mais le calcul conduit bien au polynôme de spin suivant :

$$G^{(3)}_{(6,6,6),(4,1,1)}(q) \;=\; q^2 + 2q^3 + 3q^4 + 2q^5 + q^6 \quad .$$

Voici donc les 9 tableaux de rubans solutions de notre problème. Nous les présentons en 3 lignes, chaque ligne correspondant à une des trois branches partant de α dans l'arbre d'exécution complet précédent lu de gauche à droite. Par ailleurs, nous avons colorié les formes constituées des rubans numérotés 1 communes à plusieurs tableaux de rubans :



Trois formes sont ainsi chacune communes à deux tableaux de rubans, expliquant par là la structure du graphe d'exécution par rapport à l'arbre d'exécution. L'idée sous-jacente est simplement que partant d'une forme donnée, plusieurs étapes de mise en place de rubans numérotés de r à i peuvent conduire à la même sous-forme qui devra être pavée de rubans numérotés de $i-1$ à 1.

Références bibliographiques

[1] A. Betten, S. Veigneau, and A. Wasserman. Parallel computation in Combinatorics using PVM. *Electr. J. of Discr. Math. and Theor. Comp. Sci.*, (1996). soumis.

[2] A.N. Kirillov, A. Lascoux, B. Leclerc, and J.-Y. Thibon. Séries génératrices pour les tableaux de dominos. *Comptes Rendus de l'Académie des Sciences*, 318: 395–400, (1994).

[3] D.E. Knuth. Permutations, matrices and generalized Young tableaux. *Pacific J. Math.*, 34: 709–727, (1970).

[4] D.E. Knuth. *The art of computer programming*, volume 3. Addison-Wesley, (1985).

[5] A. Lascoux. Cyclic permutations on words, tableaux and harmonic polynomials. In Manoj Prakashan, editor, *Proc. Hyderabad Conference on Algebraic Groups*, Madras, (1991).

[6] A. Lascoux, B. Leclerc, and J.-Y. Thibon. Ribbon tableaux, Hall-Littlewood functions, quantum affine algebras and unipotent varieties. *Journal of Mathematical Physics*, (1995). soumis.

[7] A. Lascoux and M.P. Schützenberger. Arêtes et tableaux. In *Proc. 20-ième séminaire Lotharinghien de Combinatoire*, pages 109–120, (1988).

[8] A. Lascoux and M.P. Schützenberger. Keys and standard bases, in invariant theory and tableaux, (1990).

[9] B. Leclerc and J.-Y. Thibon. Canonical bases of q-deformed Fock spaces. *International Math. Research Notices*, (1996). à paraître.

[10] C. Schensted. Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13: 179–191, (1961).

[11] D. Stanton. *Invariant theory and tableaux.* Springer-Verlag, New York, (1986).

[12] D. Stanton and D. White. A Schensted algorithm for rim-hook tableaux. *J. Comb. Theory A*, 40: 211–247, (1985).

[13] S. Veigneau. Distributed computation of ribbon tableaux and spin polynomials. In A. Bode, J. Dongara, T. Ludwig, and V. Sunderam, editors, *Proc. 3rd European PVM User's Group Meeting*, volume 1156, Munich, (1996). Springer-Verlag.

8

Parallélisation et analyse de performance

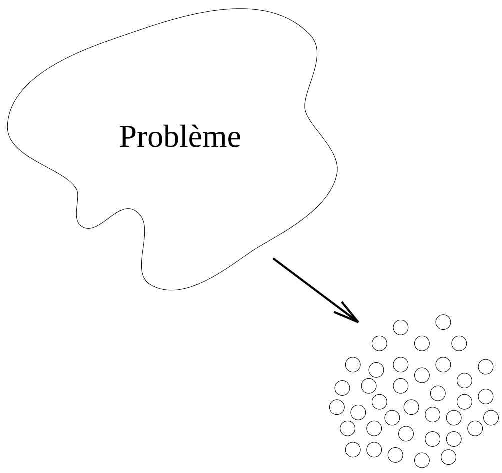
On ne comprend pas le logiciel,
et parfois on ne comprend pas la machine non plus,
mais on peut "voir" des lampes qui clignotent quand même.

La conception d'un programme parallèle peut tirer profit d'une mise en œuvre méthodique et réfléchie. Les mécanismes mis en jeu sont suffisamment complexes pour qu'il soit plutôt difficile de trouver le bon compromis entre calculs et communications ou encore entre synchronisme et asynchronisme par exemple. La solution de parallélisation d'un algorithme n'est dans beaucoup de cas pas unique ne serait-ce que par la diversité des algorithmes séquentiels apportant une solution au problème et servant souvent de point de départ de l'algorithmique parallèle. Ce paragraphe décrit la parallélisation de l'algorithme d'énumération décrit précédemment en essayant de dégager les étapes principales du processus de parallélisation. Cette partie rentre dans le cadre d'une première étude des possibilités de parallélisation d'algorithmes en combinatoire ; elle vise en fait à dessiner un axe de travail futur pour la mise en place d'un logiciel de calcul symbolique utilisant des ressources distribuées et portant essentiellement sur la manipulation de structures combinatoires utilisées en combinatoire algébrique.

8.1 Distribution du calcul du polynôme de spin

Nous avons étudié la parallélisation de l'algorithme d'énumération de l'ensemble des tableaux de rubans d'une forme et d'un poids donnés associé au calcul du polynôme de spin correspondant. Plus exactement, nous avons envisagé la distribution du calcul des polynômes de spin sur plusieurs ordinateurs regroupés au sein d'une machine parallèle virtuelle décrite via le logiciel `PVM`.

Une première étape dans la parallélisation d'un algorithme est souvent le découpage du problème en opérations élémentaires, l'idée force étant de dégager les tâches pouvant être exécutées en parallèle en ignorant toute considération pratique concernant par exemple

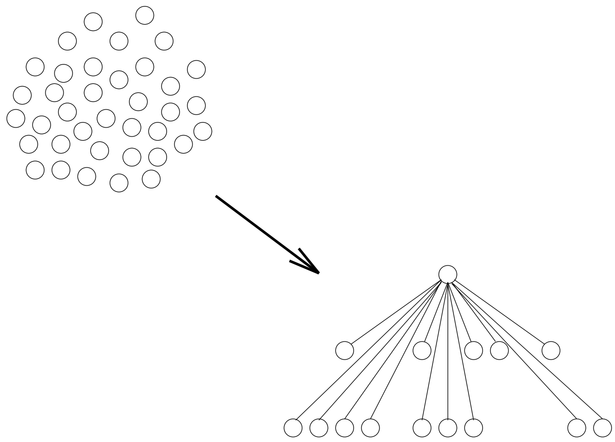


le nombre de processeurs qui seront utilisés. Bref, l'accent est mis sur les opportunités de concurrence entre opérations élémentaires. Deux choix de découpage sont essentiellement possibles, l'un basé sur l'étude des données manipulées, l'autre s'appuyant en premier sur la décomposition du calcul lui-même. En ce qui concerne l'énumération des tableaux de rubans, nous avons choisi de diviser le calcul (et non pas les données) en deux parties, l'une étant administrée par ce que l'on appellera désormais le *processus maître*, l'autre partie étant prise en charge par l'ensemble des *processus esclaves*, le processus maître et les processus esclaves étant des tâches, c'est-à-dire essentiellement des processus au sein de la machine parallèle virtuelle.

D'un côté, le processus maître initie, gère et termine le calcul du polynôme de spin. Initier le calcul signifie qu'il doit lire les données d'entrée λ, μ mais aussi le nombre maximal de processus esclaves simultanément présents sur l'ensemble des machines physiques. Nous reviendrons plus loin sur la gestion du calcul. Finalement, le processus maître termine le calcul en produisant le polynôme de spin $G^{(k)}_{\lambda,\mu}(q)$ à partir du polynôme $F^{(k)}_{\alpha,\mu}(q)$ calculé récursivement en fonction des différents résultats produits par l'ensemble des processus esclaves.

De l'autre côté, les processus esclaves envisagent toutes les possibilités. Chacun d'eux correspond en fait à un nœud du graphe d'exécution. Chacun doit ainsi énumérer toutes les permutations distinctes $v_r\tau$ of v_r—qui indicent les arêtes du graphe—et éliminer les cas trivialement non admissibles.

La deuxième étape de la parallélisation d'un algorithme concerne cette fois la communication qui doit être établie entre toutes les tâches participant au calcul. Cette phase explique globalement quelle va être la nature des messages nécessaires à la synchronisation des intervenants. Nous continuons la description de la parallélisation de notre algorithme en abordant ce point précis de la communication. Comme nous l'avons dit plus haut, le processus maître doit aussi gérer le calcul du polynôme de spin, *i.e.* accepter ou refuser les requêtes de création de nouvelles tâches. Chaque requête contiendra plusieurs demandes de création de tâches contenues dans un message et la réponse correspondra aussi à un seul message. Ces requêtes viennent des processus esclaves qui, après avoir énuméré et stocké des vecteurs valides, demandent au processus maître s'ils peuvent créer les nouvelles tâches correspondant à ces vecteurs. En général, les processus esclaves créeront les nouvelles tâches acceptées avec leurs paramètres puis finalement informeront le processus maître de leur terminaison.



Avant de continuer, revenons quelques lignes plus haut lorsque nous parlions de la gestion du calcul. Nous devons préciser le comportement du processus maître dont le rôle est bien entendu primordial. Après avoir lu les données d'entrée, il crée le premier esclave qui correspond au sommet du graphe indicé par α. Puis, il rentre dans une boucle `while` qui ne terminera que lorsqu'il n'y aura plus de tâches œuvrant à la construction du polynôme. Le processus maître est en fait en attente d'un message. Ce peut être un message l'informant qu'une tâche du dernier niveau du graphe d'exécution amène une solution au problème, ou qu'une tâche a tout simplement terminé son travail. Comme nous l'avons dit plus haut, le processus maître reçoit aussi des demandes de créations de nouvelles tâches et la réponse à ces demandes dépend de plusieurs paramètres. Par exemple, le processus maître doit vérifier que l'on n'a pas atteint le nombre de processus maximal autorisé sur la machine virtuelle et adapter sa réponse en fonction. Dans le cas où ce nombre maximal est atteint, il demandera à la tâche émettrice du message de continuer le calcul en séquentiel jusqu'à ce qu'une ou plusieurs "places" sur la machine parallèle soient disponibles. L'objet de cette limitation du nombre de processus (au sens système) travaillant à la fois est double : d'abord l'algorithme d'énumération est très vorace en nombre de tâches élémentaires et il n'est donc pas envisageable de ne mettre aucune limite sous peine de voir les performances de la machine s'écrouler sous le nombre de processus et puis cela présente l'avantage d'une simulation sur une machine parallèle disposant d'un nombre fixé de processeurs. Nous verrons dans le chapitre 8 que ce point de vue a son importance au niveau de l'analyse et de la visualisation de l'exécution du programme.

Comme nous l'avons précisé dans la description de l'algorithme, il est tout à fait possible qu'à un même niveau de l'arbre d'exécution nous trouvions des vecteurs triés identiques. De fait, la structure d'exécution a l'aspect d'un graphe et c'est pourquoi le processus maître doit faire en sorte qu'un sous-arbre (ou sous-graphe) donné ne soit calculé qu'une seule fois et ce même si plusieurs tâches ont émis les mêmes requêtes de calcul.

Remarque. Cela reste vrai dans tous les cas de figures, c'est-à-dire non seulement lors de la création de nouvelles tâches mais aussi lorsqu'une tâche poursuit un calcul en séquentiel. Par ailleurs, le processus maître ordonne des appels récursifs plutôt que des créations de tâches lorsque la machine est surchargée mais cela devient transparent pour lui au niveau de la communication. Le processus maître n'est plus capable par la suite de dire si un message qui lui parvient provient d'un appel récursif ou d'une nouvelle tâche. La programmation des processus esclaves est donc totalement indépendante de celle du processus maître ; les deux programmes peuvent donc subir des modifications profondes du moment que le protocole de dialogue est respecté.

Le processus maître compte simplement le nombre de tâches effectivement lancées (que l'on distingue du nombre de tâches élémentaires mises à contribution pour la construction du polynôme) et se termine en produisant le résultat lorsque toutes les tâches ont terminé leur travail.

Juste avant la mise en application des principes décrits ci-dessus, une phase de regroupement des opérations élémentaires dégagées au cours de la première étape de développement, a pour objet de faire le point sur les ressources disponibles (physiquement) et les coûts de développements par rapport aux performances escomptées. Cette étape est très importante car elle conduit à des choix décisifs de programmation : il faut essayer de minimiser les communications et favoriser la concurrence. La chose n'est pas facile sur une machine parallèle et elle l'est encore moins lorsque l'on envisage une distribution des calculs sur des machines hétérogènes. Cela rejoint la dernière étape où l'on associe à chaque tâche un processeur chargé de la réaliser. Faut-il gérer la distribution en fonction des spécificités des machines ? La question est digne d'intérêt car les fondements même du programme en seront bouleversés. Ou bien l'on admet que le domaine de la répartition des tâches sur les différents ordinateurs est du ressort de `PVM` par exemple, ou alors on intègre un algorithme qui sera le plus souvent spécifique au problème à résoudre. Dans notre cas nous avons choisi de laisser `PVM` apprécier quelles seront les machines sur lesquelles les nouvelles tâches seront lancées.

8.2 Quelques résultats numériques

Les essais que nous avons pu faire montrent clairement à la fois les possibilités et les limitations du programme et de la machine virtuelle. De précédents calculs avaient été effectués grâce à un programme écrit pour le système Maple, c'est pourquoi nous avions un ordre d'idée de la taille des données d'entrée conduisant à un résultat accessible, le "accessible" faisant référence à des calculs de quelques heures. Il est cependant difficile de comparer les temps d'exécution d'un programme C et d'un programme Maple car le programme C est par nature plus rapide. Les premiers calculs que nous avons fait nous ont permis de donné un polynôme de spin en utilisant 8 ordinateurs en plus ou moins une minute alors que le programme Maple produisait ce même résultat en 2 à 3 heures. Cet exemple peut être pris comme une sorte de référence car nous sommes désormais effectivement capables de produire des polynômes de spin nettement plus "gros" en énumérant plusieurs millions et même milliards de tableaux de rubans.

Nous présentons quelques exemples d'exécution de notre programme dont le processus maître s'appelle `ribbon`. L'utilisateur a la possibilité de demander via la ligne de commande certaines options telles que :

- `-data` affiche les données d'entrée ;
- `-tasks` fixe le nombre maximal de processus (au sens système) physiquement présents sur l'ensemble de la machine virtuelle. Par défaut, ce nombre est égal à 10 ;
- `-nb` affiche le nombre de solutions, c'est-à-dire pose $q = 1$ dans le polynôme de spin ;
- `-spinpoly` affiche le polynôme de spin ;
- `-execstats` présente quelques statistiques d'exécution ;
- `-graphstats` donne des informations sur le graphe d'exécution.

Les différents essais que nous avons effectués nous ont permis de caractériser plusieurs propriétés impliquant à la fois la nature de la machine virtuelle et la nature des tâches s'exécutant sur cet ensemble de ressources. Notre premier exemple de référence est le calcul des 34990795 tableaux de rubans de forme (12^8) et d'évaluation (4^6). Nous avons lancé plusieurs fois notre programme `ribbon` en changeant le nombre de processeurs constituant la machine virtuelle et le nombre de tâches (processus) lancées simultanément. Pour essayer d'appréhender les performances globales de la machine virtuelle, nous avons distingué deux blocs principaux fournisseurs de ressources. Le premier bloc que nous appellerons désormais A est constitué de machines affichant des performances limitées (Silicon Graphics de type INDY ou INDIGO, stations NeXT, etc.) alors que le deuxième ensemble B de machines regroupe des ordinateurs plus puissants, essentiellement de type serveur (HP9000, Silicon Graphics ONYX)

Par exemple dans l'exemple suivant, nous avons réalisé un test impliquant une vingtaine de processeurs parmi les deux groupes A et B. La commande UNIX `timex` est utilisée pour afficher le temps d'exécution du processus maître alors que l'option `-execstats` produit un résultat concernant le cumul des différents temps d'exécutions des processus esclaves répartis sur l'ensemble de la machine virtuelle :

```
$ timex ribbon -tasks 10 -data -nb -execstats -labels
Input data          : 8 12 12 12 12 12 12 12 12 6 4 4 4 4 4 4
Number of solutions : 34990795

Execution statistics:
     total user time: 45.02 sec.
      total sys time: 18.99 sec.
    master user time: 20.00 sec.
     master sys time: 2.38 sec.
     tasks user time: 25.02 sec.
      tasks sys time: 16.61 sec.
      total nb tasks: 2452
       total spawned: 295
   max spawned tasks: 10
        max nb tasks: 698

real       54.06
user       20.52
sys         2.44
```

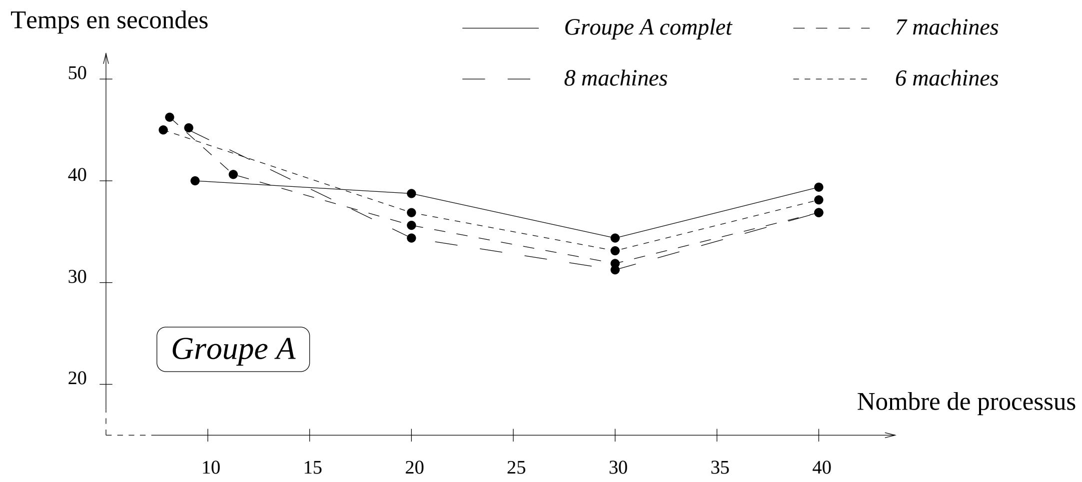
A noter que le format d'entrée des formes et poids des tableaux de rubans correspond à la suite de nombres $|\lambda|\ \lambda_1\ \lambda_2\ \ldots\ \lambda_{|\lambda|}|\mu|\ \mu_1\ \mu_2\ \ldots\ \mu_{|\mu|}$. Au total, il nous a fallut 2452 opérations élémentaires, *i.e.* 2452 sommets dans le graphe d'exécution, pour calculer le résultat parmi lesquelles seulement 169 ont été effectivement lancées, les autres correspondant à des appels récursifs.

Notons que le processus maître a bien joué son rôle en limitant à 10 le nombre de processus simultanés sur la machine virtuelle, ce nombre ayant été atteint pendant l'exécution. Nous produisons une autre information intéressante concernant le nombre maximal d'opérations élémentaires en cours, c'est-à-dire acceptées par le processus maître. Ce nombre (698 dans l'exemple précédent) donne une information sur la charge globale de la machine virtuelle.

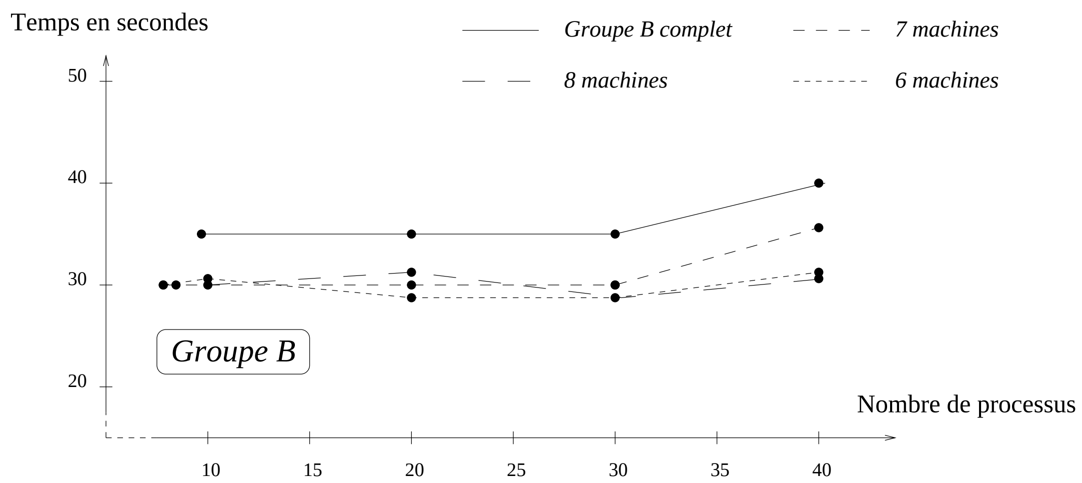
Nous présentons maintenant le même calcul en faisant varier le nombre maximal de processus simultanés afin de montrer l'impact de ce nombre sur le temps réel d'exécution :

```
$ timex ribbon -tasks 15 -execstats -labels
Execution statistics:
     total user time: 32.63 sec.
      total sys time: 12.88 sec.
    master user time: 20.14 sec.
     master sys time: 1.85 sec.
     tasks user time: 12.49 sec.
      tasks sys time: 11.03 sec.
      total nb tasks: 2452
       total spawned: 206
   max spawned tasks: 15
        max nb tasks: 997

real       47.15
user       20.65
sys         1.90

$ timex ribbon -tasks 20 -execstats -labels
Execution statistics:
     total user time: 37.11 sec.
      total sys time: 11.88 sec.
    master user time: 20.60 sec.
     master sys time: 2.21 sec.
     tasks user time: 16.51 sec.
      tasks sys time: 9.67 sec.
      total nb tasks: 2452
       total spawned: 169
   max spawned tasks: 20
        max nb tasks: 1121

real       28.94
user       21.12
sys         2.27

$ timex ribbon -tasks 25 -execstats -labels
Execution statistics:
     total user time: 38.73 sec.
      total sys time: 8.61 sec.
    master user time: 20.09 sec.
     master sys time: 1.83 sec.
     tasks user time: 18.64 sec.
      tasks sys time: 6.78 sec.
      total nb tasks: 2452
       total spawned: 89
   max spawned tasks: 25
        max nb tasks: 1335

real       26.48
user       20.61
sys         1.88
```

Ces quelques essais montrent que le nombre de tâches simultanées que l'on peut autoriser ne correspond pas nécessairement au nombre de machines impliquées dans la machine virtuelle. Dans le cas du calcul de la constante π dont nous avons parlé précédemment, les tâches prenaient un temps suffisamment long pour que le coût de la communication avec l'administrateur du calcul ne rentre absolument pas en jeu dans l'évaluation du temps global d'exécution. Dans l'exemple précis de l'énumération des tableaux de rubans, le problème est complètement différent puisque les tâches créées sont suffisamment élémentaires pour que le coût de la création de la tâche et de la communication avec le processus maître représentent une part très importante du temps d'exécution global. A ce sujet, nous avons effectué plusieurs essais de configuration de machines afin de mettre en évidence cette propriété de la machine virtuelle. Pour chaque configuration, nous avons fait varier le nombre de processus pouvant exister simultanément. Nous présentons ci-dessous les résultats obtenus, tout d'abord pour le groupe A :



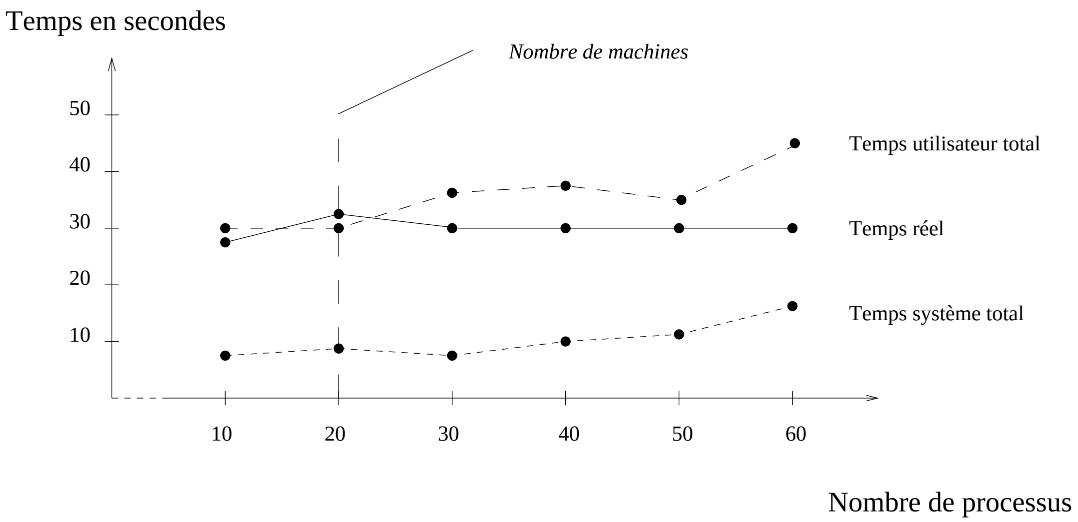
puis pour le groupe B :



Les allures de ces différentes courbes, aussi bien dans le groupe A que dans le groupe B, montrent que 20, 30 voire 40 processus peuvent simultanément exister sur la machine virtuelle sans pour autant altérer les performances globales de la machine virtuelle. Ceci est très spécifique à notre programme d'énumération des tableaux de rubans et cela s'explique par le fait que les tâches passent beaucoup de temps à attendre des autorisations de calcul de la part du processus maître. Du coup, il semble qu'il soit intéressant de placer plusieurs tâches sur un même processeur afin qu'une tâche en attente ne corresponde pas à un processeur inactif mais que potentiellement une autre tâche continue son travail sur le processeur.

Si nous fixons à 30 le nombre de processus simultanés pour des configurations de 6 à 20 machines prises parmi les groupes A et B réunis, nous obtenons des temps d'exécution oscillant entre 30 et 35 secondes. Ces temps sont à comparer avec les 40 secondes (respectivement les 43 secondes) nécessaires à l'énumération séquentielle des solutions sur une machine type du groupe A (resp. B). Cette comparaison fait apparaître que l'augmentation du nombre de machines ne s'est pas répercutée sur la performance du programme distribué. Nous avons eu un gain de près de 15 secondes qui représentent de l'ordre de 30% des temps séquentiels constatés. Pourtant, le nombre de machines n'y est pour rien tant la communication entre les processus prend du temps.

La figure suivante présente trois courbes, l'une étant le temps réel pris par le calcul *i.e.* celui du processus maître, la deuxième le temps utilisateur (temps pendant lequel on dispose des ressources du processeur) cumulé de l'ensemble des processus esclaves et la troisième courbe le temps système cumulé de ce même ensemble de processus. Le calcul a été réalisé sur 20 machines :



Ce graphique montre que le temps réel de calcul est constant alors que le temps utilisateur total a une légère tendance à augmenter avec le nombre de processus, tout comme le temps système total. La répartition des tâches par `PVM` a une grande influence sur l'allure des différentes courbes mais globalement, le fait que la courbe du temps réel soit sous la courbe du temps utilisateur montre qu'il y a un faible gain dans la distribution de ce calcul.

Nous avons augmenté le poids de la partition λ pour passer de $8 \times 12 = 96$ à un poids de $8 \times 15 = 120$. Le calcul demande alors plus de ressources comme l'illustrent les données suivantes qui correspondent à l'énumération des 194014660 tableaux de rubans de forme $\lambda = (15, 15, 15, 15, 15, 15, 15, 15)$ et de poids $\mu = (4, 4, 4, 4, 4, 4)$:

```
$ timex ribbon -tasks 25 -nb -execstats -labels
Number of solutions : 194014660

Execution statistics:
     total user time: 166.18 sec.
      total sys time: 28.60 sec.
    master user time: 73.13 sec.
     master sys time: 4.12 sec.
     tasks user time: 93.05 sec.
      tasks sys time: 24.48 sec.
      total nb tasks: 3947
       total spawned: 446
   max spawned tasks: 25
        max nb tasks: 1960

real     1:38.63
user     1:14.75
sys         4.23
```

Passons finalement à un exemple plus conséquent. Il s'agit de la mise en place des 24 6-rubans nécessaires au pavage de la forme $\lambda = (18, 18, 18, 18, 18, 18, 18, 18)$. Nous gardons encore le même poids $\mu = (4, 4, 4, 4, 4, 4)$ et obtenons plus d'un milliard de solutions :

```
$ ribbon -tasks 30 -nb -labels -data -spinpoly -graphstats
Input data          : 8 18 18 18 18 18 18 18 18 6 4 4 4 4 4 4
Number of solutions : 1079706100
Spin polynomial     : 5*q^6+75*q^7+554*q^8+2835*q^9+
                      11530*q^10+39470*q^11+117927*q^12+314030*q^13+
                      756571*q^14+1665504*q^15+3375529*q^16+6332801*q^17+
                      11046379*q^18+17976562*q^19+27375288*q^20+
                      39109524*q^21+52541166*q^22+66516023*q^23+
                      79511677*q^24+89908914*q^25+96335442*q^26+
                      97959748*q^27+94667092*q^28+87048607*q^29+
                      76241169*q^30+63654091*q^31+50690602*q^32+
                      38513325*q^33+27917667*q^34+19299417*q^35+
                      12713296*q^36+7968919*q^37+4743509*q^38+
                      2673314*q^39+1420871*q^40+708146*q^41+
                      328508*q^42+140270*q^43+54314*q^44+18589*q^45+
                      5424*q^46+1236*q^47+180*q^48

Graph statistics    :
              height: 6
          level info: (1/1) (120/700) (1531/2800) (3006/5600)
                      (1531/2800) (165/700)
     number of nodes: 6354
    max nb daughters: 120 239 211 113 36 1
```

Le polynôme de spin nous indique par exemple qu'il y a 94667092 tableaux de spin 28 parmi les 1079706100 solutions. Nous produisons en même temps des statistiques sur le graphe d'exécution. Ces informations permettent de visualiser la structure générale du graphe. Dans l'exemple précédent, le graphe a 6354 sommets et la répartition des sommets sur chaque niveau est la suivante :

```
         level info: (1/1) (120/700) (1531/2800) (3006/5600)
                     (1531/2800) (165/700)
   max nb daughters: 120 239 211 113 36 1
```

La ligne `level info` donne pour les 4 étages du graphe, le nombre de sommets utilisés par rapport au nombre de sommets alloués. La ligne `max nb daughters` donne le nombre maximal de successeurs d'un sommet de chacun des niveaux. Le 1 final rappelle notamment qu'il y a au plus une solution pour retomber sur une permutation de ρ_s en partant d'un vecteur de l'avant dernier niveau. Cette information est intéressante puisqu'elle correspond au nombre de branches valides partant au maximum d'un sommet donné.

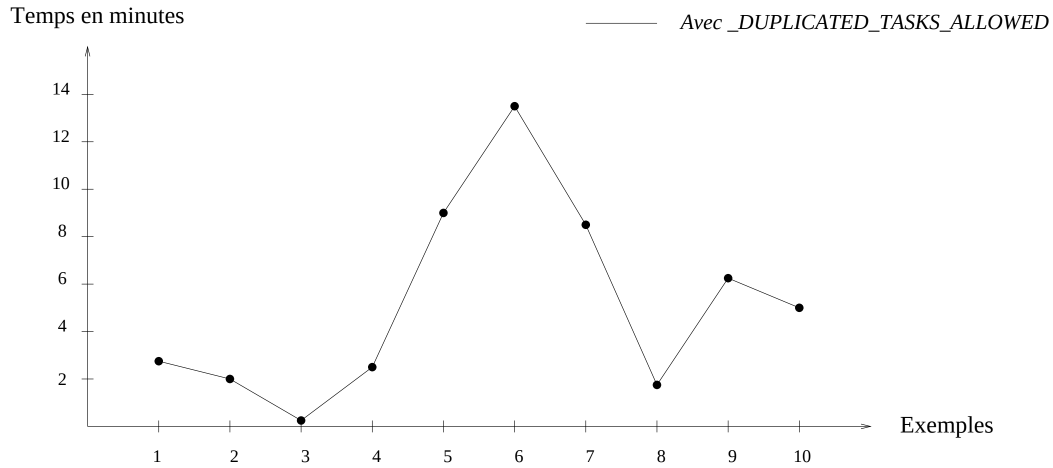
Nous avons réalisé plusieurs essais sur 10 machines du groupe B en faisant varier le nombre de processus simultanés de 30 à 60. Les temps vont de 6 minutes à 8 minutes alors qu'il faut 9 minutes de calcul en séquentiel. Les 6 minutes sont atteintes dans plusieurs cas correspondant vraisemblablement à une bonne répartition des tâches sur le parc de machines.

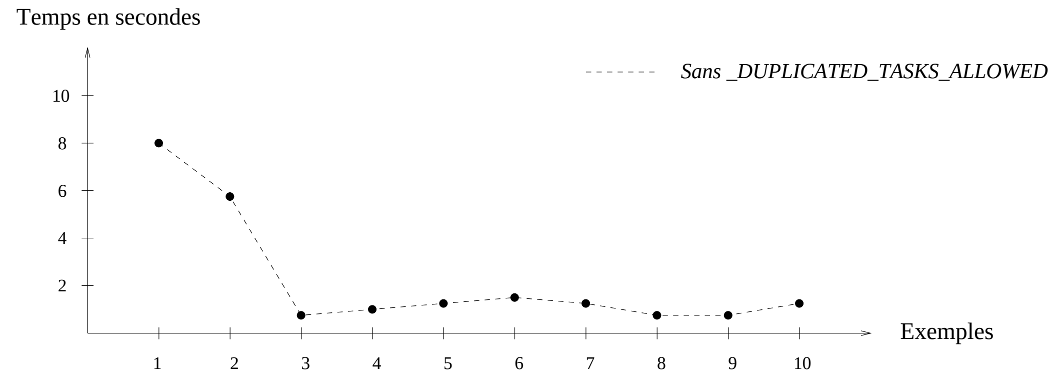
Nous avons aussi essayé d'utiliser une seule machine très puissante pour simuler une machine parallèle à 20 processeurs.

```
$ timex ribbon -tasks 20 -labels -execstats
Execution statistics:
     total user time: 114.71 sec.
      total sys time: 24.33 sec.
    master user time: 57.23 sec.
     master sys time: 3.89 sec.
     tasks user time: 57.48 sec.
      tasks sys time: 20.44 sec.
      total nb tasks: 6353
       total spawned: 1209
   max spawned tasks: 20
        max nb tasks: 1971

real     6:13.63
user       58.32
sys         3.89
```

Le temps utilisateur total est plus faible que dans le cas d'un ensemble de processeurs (114 secondes contre 600) grâce à la puissance de l'unique machine employée mais le temps total reste équivalent au temps constaté en réelle distribution du calcul.

Nous avons signalé dans le chapitre précédent que l'arbre d'exécution était en réalité un graphe en ce sens que plusieurs branches se rejoignent. Nous avons donc intégré à notre programme `ribbon` l'option `_DUPLICATED_TASKS_ALLOWED` permettant de prendre ou de ne pas prendre en compte cette propriété. Lorsque `_DUPLICATED_TASKS_ALLOWED` est positionné, le processus maître répondra toujours positivement aux demandes de création de tâches provenant des processus esclaves, dans la limite du nombre maximum de processus simultanés. Nous donnons ci-dessous un graphique comparant les temps séquentiels pour 12 exemples significatifs calculés avec ou sans l'option `_DUPLICATED_TASKS_ALLOWED` sur 8 machines du groupe B et avec un nombre maximal de processus simultanés fixé à 10 :

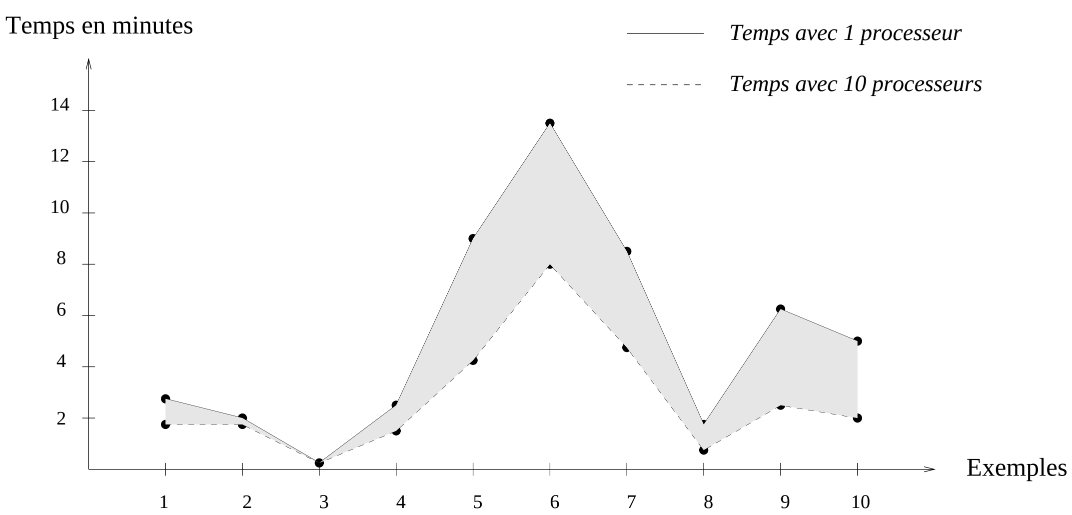




Ces courbes montrent clairement que cette propriété de l'arbre d'énumération des tableaux de rubans est à la base de la performance de notre programme. Cela nous a permis de calculer de très gros polynômes de spin. Par exemple, il y a 12429633862800 tableaux de rubans de forme (18^8) et d'évaluation (2^{12}). Cet ensemble correspond au polynôme de spin $462 + 10890 * q + 126390 * q^2 + 972796 * q^3 + 5622925 * q^4 + 26142413 * q^5 + 102113473 * q^6 + 345274997 * q^7 + 1032530114 * q^8 + 2774049915 * q^9 + 6773987934 * q^{10} + 15165318870 * q^{11} + 31329858350 * q^{12} + 60021862133 * q^{13} + 107044013148 * q^{14} + 178251016804 * q^{15} + 277842395664 * q^{16} + 406243379216 * q^{17} + 558237900853 * q^{18} + 722198797224 * q^{19} + 881077135626 * q^{20} + 1015247904942 * q^{21} + 1106575205996 * q^{22} + 1142498961276 * q^{23} + 1118850943106 * q^{24} + 1040538638745 * q^{25} + 919995532475 * q^{26} + 774035874881 * q^{27} + 620175672491 * q^{28} + 473466722770 * q^{29} + 344530359290 * q^{30} + 238976321594 * q^{31} + 157965829121 * q^{32} + 99445870580 * q^{33} + 59562846761 * q^{34} + 33889029328 * q^{35} + 18276953849 *$

$q^{36} + 9316497124 * q^{37} + 4471453613 * q^{38} + 2010593948 * q^{39} + 841485865 * q^{40} + 324995682 * q^{41} + 114501519 * q^{42} + 36219909 * q^{43} + 10056585 * q^{44} + 2368190 * q^{45} + 447348 * q^{46} + 60973 * q^{47} + 4642 * q^{48}$. Ce polynôme a été calculé en 5 minutes sur un seul processeur et pourtant il nous aura suffi de 2 minutes pour calculer notre plus gros polynôme de spin $5830160336 + 66808878136 * q + 384231341914 * q^2 + 1483104446782 * q^3 + 4327455304766 * q^4 + 10182537350474 * q^5 + 20118970369726 * q^6 + 34315557018574 * q^7 + 51557356550780 * q^8 + 69309776303574 * q^9 + 84433833356033 * q^{10} + 94213315485587 * q^{11} + 97189925319873 * q^{12} + 93454994187688 * q^{13} + 84373925542153 * q^{14} + 71981642566337 * q^{15} + 58354876062971 * q^{16} + 45172242286462 * q^{17} + 33525647454527 * q^{18} + 23936228311917 * q^{19} + 16484646427767 * q^{20} + 10973763455060 * q^{21} + 7072118526393 * q^{22} + 4416824656145 * q^{23} + 2674790753795 * q^{24} + 1570951420847 * q^{25} + 894626066983 * q^{26} + 493732508699 * q^{27} + 263842132586 * q^{28} + 136365302784 * q^{29} + 68069734457 * q^{30} + 32760723625 * q^{31} + 15171443081 * q^{32} + 6744455324 * q^{33} + 2870163928 * q^{34} + 1165419704 * q^{35} + 449760766 * q^{36} + 164200702 * q^{37} + 56390020 * q^{38} + 18090076 * q^{39} + 5374228 * q^{40} + 1462273 * q^{41} + 359191 * q^{42} + 78131 * q^{43} + 14650 * q^{44} + 2276 * q^{45} + 275 * q^{46} + 23 * q^{47} + q^{48}$, lequel correspond aux 923497397222400 tableaux de rubans de forme $(36, 36, 18, 18, 18, 18)$ et d'évaluation (1^{24}).

Finalement, l'étude de la distribution de l'énumération des tableaux de rubans a été faussée par la nature du graphe d'exécution qui fait que le temps de calcul n'est pas une fonction croissante de la taille des données d'entrée. De fait, les temps d'exécution en distribué ne montrent pas clairement l'intérêt du parallélisme. Par contre, si le processus maître décide que tous les chemins de l'arbre d'exécution doivent être effectivement envisagés (par utilisation de l'option `_DUPLICATED_TASKS_ALLOWED`), alors les temps d'exécution seront plus directement liés au nombre de solutions. C'est pourquoi nous présentons un dernier graphique permettant de comparer les temps séquentiels avec les temps obtenus sur 10 processeurs du groupe B avec un nombre maximal de 30 processus simultanés pour les mêmes exemples du graphique précédent :



La zone délimitée montre que la distribution du calcul a apporté un gain en temps de l'ordre de 40% pour les énumérations correspondant à un nombre important de solutions.

8.3 Analyse de performance en environnement PVM

La performance globale d'une machine parallèle est directement en relation avec la puissance ou la nature des processeurs qui la composent, la mémoire dont dispose chaque processeur ou la mémoire globale, mais elle est aussi fortement dépendante du réseau d'interconnexion qui relie l'ensemble des processeurs. Si l'on veut envisager d'atteindre des performances 1000 fois supérieures aux performances des superordinateurs, il faut revoir les principes architecturaux des ordinateurs. Ces architectures ont considérablement évolué grâce notamment à l'augmentation des capacités d'intégration, mais ce n'est pas suffisant pour mettre en œuvre les puissances de calculs demandées. C'est pourquoi les recherches s'orientent désormais vers des machines ayant des milliers de processeurs aptes à communiquer très rapidement entre eux. Cette communication est perçue comme la clé de la performance. L'architecture du réseau d'interconnexion d'une machine parallèle à mémoire distribuée est un des points les plus fondamentaux pour sa programmation et son efficacité. La topologie choisie fait que les machines sont plus ou moins différentes les unes des autres et que, de ce fait, le coût d'échanges de messages est plus ou moins important. Cette communication interprocesseurs est à la base même du parallélisme, du fait des échanges entre mémoires. Une communication peu efficace ou mal gérée influence fortement les performances que l'on pourrait espérer d'une machine donnée. Mais ce n'est pas tout car il faut aussi considérer le rapport entre coût de calcul et coût de communication : il est inutile d'avoir des processeurs puissants si le réseau d'interconnexion est incapable de fournir à temps les données nécessaires aux calculs.

Ce sont tous ces paramètres que l'on devrait maîtriser pour espérer obtenir la performance maximale d'une machine parallèle donnée. Les constructeurs travaillent ainsi à la réalisation de machines à architectures en grille torique, tore 3D ou encore réseaux multi-étages. Par exemple, Cray Research Inc. s'intéresse désormais aux machines parallèles vectorielles à mémoire partagée alors que toute sa renommée s'est fondée sur le développement des machines mono-processeur les plus puissantes du marché. Par exemple, le Cray MPP ou T3D dispose d'une architecture pouvant accueillir plusieurs centaines voire plusieurs milliers de processeurs Dec Alpha, interconnectés sous forme d'un tore 3D, d'où son nom. En fait, tous espèrent pouvoir afficher de très hautes performances en faisant en sorte que l'accès aux données ne soit pas un goulot d'étranglement.

Les mêmes principes s'appliquent bien sûr aux machines parallèles virtuelles. Cependant, alors que dans le cas d'une machine parallèle ces problèmes sont résolus au cours de la réalisation de la dite machine, dans le cas de la mise en place d'une machine parallèle virtuelle, c'est souvent à l'utilisateur d'envisager une solution logicielle à ces problèmes puisqu'il ne peut pas vraiment contrôler l'architecture du réseau d'interconnexion par exemple. Cette gestion logicielle est un point douloureux tant il est difficile d'en apprécier l'influence sur la performance globale d'un programme.

En fin de compte, le travail du programmeur de machines parallèles est somme toute très proche du travail de programmation d'ordinateurs séquentiels à ceci près qu'une notion de placement des tâches apparaît et que le traçage d'exécution n'est plus aussi facile. Nous avons vu au chapitre 7 comment s'organisait le développement d'un programme parallèle et notamment nous avons mis en évidence le problème de placement des tâches. Si on oublie ce problème qui peut être pris en compte par l'environnement dans lequel va s'exécuter une application donnée, il reste à traiter la problématique du traçage de l'exécution de programmes parallèles. Un algorithme séquentiel à vocation purement calculatoire (par opposition aux applications systèmes) est la plupart du temps déterministe. Etant donné une donnée d'entrée, la suite d'instructions exécutée est toujours la même ou plus exactement, l'ordre dans lequel cette suite d'instructions est exécutée est toujours la même. Par conséquent, les résultats obtenus sont tous identiques et ce pour toutes les exécutions séquentielles du programme. Même si l'objectif est vraisemblablement le même dans le cas de programmes distribués, c'est-à-dire obtenir un résultat unique pour toutes les exécutions du programme parallèle, on ne peut plus dire que l'ordre des instructions exécutées est identique pour toutes les exécutions. Ce non déterminisme s'affiche à deux niveaux. Le premier niveau concerne la synchronisation des processus. Par exemple, les langages de programmation des machines distribuées fournissent en général des fonctions permettant de tester l'arrivée d'un message sur une voie de communication. D'une exécution à l'autre, les messages sont susceptibles d'arriver dans un ordre différent ce qui entraîne des ordres différents d'exécution des instructions du programme. L'autre niveau de non déterminisme provient du fait que les tâches sont réparties sur des machines dont les vitesses diffèrent. Le placement des tâches influence donc directement mais de manière implicite l'ordre d'exécution des instructions.

L'éventualité même de suivi de deux chemins d'instructions différents est un point critique de la phase de mise au point d'un programme distribué. En particulier, cette éventualité implique qu'une erreur survenant lors d'une exécution ne pourra pas systématiquement être reproduite lors d'une autre exécution. Il est donc nécessaire de disposer d'un moyen permettant d'assurer la similarité des exécutions sur une même donnée d'entrée. L'idée est de réaliser une exécution de référence afin de collecter des traces d'exécution, puis d'utiliser ces traces afin de reproduire l'exécution de référence. La collecte des traces, très simple en séquentiel, pose un réel problème en distribué.

Deux phases principales sont à la base des outils d'analyse de performance. La première phase dont nous venons de parler concerne la collecte des données, souvent pendant l'exécution même du programme à tracer avec parfois une visualisation temps réel. Les données stockées par exemple dans un fichier peuvent être de plusieurs natures :

① **compteurs** stockant les nombres d'occurrence d'événements ;

② **profileurs** enregistrant les temps passés dans chacune des parties d'un programme ;

③ **traces d'événements** contenant les occurrences de plusieurs types d'événements.

La seconde phase est bien sûr la phase de visualisation des données collectées. Autant la phase de collecte des données est souvent purement technique en ce sens que les problèmes rencontrés sont du domaine de la programmation, autant il semble que la phase de visualisation soit beaucoup plus difficile à appréhender. En fait, une "bonne" image d'une exécution parallèle n'est pas chose simple car une visualisation doit apporter un maximum d'information sur un nombre minimum de types de données de traçage.

Certains distinguent une troisième phase de modification des données intervenant entre les deux phases décrites précédemment. En effet, les données de traçage ne sont pas toujours au bon format pour être directement visualisées mais les modifications à apporter sont plus du domaine de la compatibilité des logiciels que du domaine du parallélisme lui-même. Il est vrai que des problèmes portant sur la reconstruction d'une horloge globale à partir des horloges des différentes machines peuvent être la cause d'un post-traitement non trivial.

Parmi les trois types de données de traçage cités plus haut, les traces d'événements sont les plus détaillées car elles peuvent inclure des informations concernant par exemple le nombre de processeurs utilisés mais surtout le type d'événement, l'adresse de destination d'un message, la taille des données échangées voire même les données échangées elles-mêmes. Ce traçage est souvent effectué dans un fichier dont la taille très importante est une réelle limitation. Par contre, cette technique apporte un recul certain sur l'exécution parallèle en mettant en avant les goulots d'étranglement en communication, les zones où le programme est passif, les moments les plus actifs en calculs et bien d'autres choses encore. Cela permet notamment de détecter des problèmes relevant de la synchronisation des processus comme par exemple un instant précis pendant lequel tous les processus émettent un message vers le même processus avec comme conséquence que sa file de messages grandissante bloque le déroulement du calcul global. Un des avantages de la richesse des fichiers de traces est qu'ils peuvent par la suite servir à générer des données de type compteur ou profileur, voire même des statistiques plus évoluées comme la taille maximale de la file de messages associée à chaque processus.

A ce propos, alors que l'environnement `PVM` semble s'imposer comme outil de base de distribution de calculs (notamment chez Cray), différentes bibliothèques d'aide à la programmation sont en cours de développement. Certaines amènent des possibilités de traçage d'exécution en insérant des instructions aux endroits du code utilisateur ou système, où l'on pourra collecter des informations utiles. C'est le cas de la bibliothèque `PGPVM`, surcouche de `PVM`, qui permet à un utilisateur d'extraire des informations sur l'exécution d'un programme donné (envoi de message, réception de message, création d'une nouvelle tâche, etc.), lesquelles informations sont collectées puis stockées dans un format lisible par le logiciel `Paragraph`. Plus précisément ce format est celui des traces générées par la librairie d'échange de messages `PICL` (*Portable Instrumented Communication Library*) qui intègre ainsi sans intervention aucune de la part du programmeur la possibilité d'extraire des traces d'exécutions. Cette phase de collecte s'accompagne en plus d'une reconstruction d'une horloge globale alors que les données collectées proviennent de machines dont les horloges ne sont *a priori* pas synchrones.

8.4 Paragraph, outil de visualisation

Développé au Oak Ridge National Laboratory, `Paragraph` est un logiciel d'analyse et de visualisation de traces d'exécution de programmes à base d'échanges de messages. Initialement programmé pour interpréter les traces générées par `PICL`, il peut très bien être utilisé pour visualiser des traces provenant d'autres librairies, si ces traces sont au bon format bien entendu. L'objet de `Paragraph` est de donner un point de vue détaillé, dynanique et visuel du comportement d'un programme parallèle ainsi qu'un résumé graphique de ses performances.

Nous donnons ci-dessous un exemple de données de traçage (à gauche) et en correspondance l'interprétation réalisée par `Paragraph` :

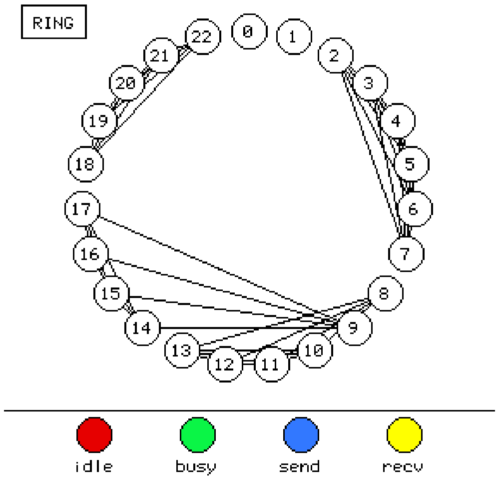
```
-3 -901 0.023579 0 -1 0                                      trace_start clock 23578 node 0
-3 -52 0.061272 0 -1 1 2 -1                                recv_blocking clock 61271 node 0
-3 -901 0.182373 1 -1 0                                     trace_start clock 182373 node 1
-3 -21 0.189360 1 -1 3 2 432 5 0    send_entry clock 189359 node 1 to 0 type 5 lth 432
```

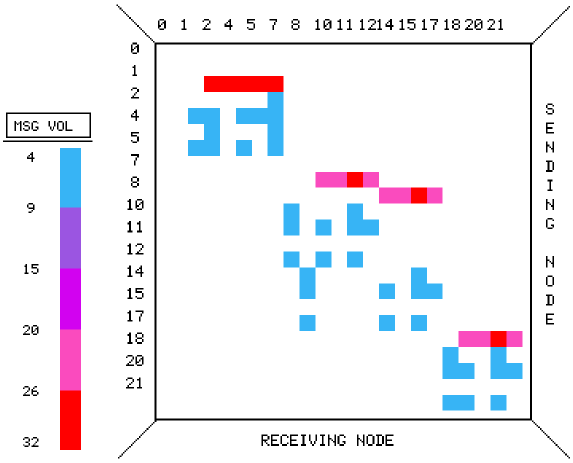
```
-4 -21 0.189726 1 -1 0                                 send exit clock 189725 node 1
-3 -52 0.189773 1 -1 1 2 -1                        recv_blocking clock 189772 node 1
-4 -52 0.190360 0 0 3 2 432 5 1     recv_waking clock 190359 node 0 from 1 type 5 lth 432
-3 -21 0.193675 0 -1 3 2 136 6 1      send entry clock 193674 node 0 to 1 type 6 lth 136
-4 -52 0.236901 1 0 3 2 136 6 0     recv_waking clock 236901 node 1 from 0 type 6 lth 136
-4 -21 0.240374 0 -1 0                                 send exit clock 240373 node 0
-3 -52 0.240422 0 -1 1 2 -1                        recv_blocking clock 240421 node 0
-3 -21 0.433116 1 -1 3 2 640 5 0      send entry clock 433115 node 1 to 0 type 5 lth 640
-4 -52 0.434116 0 0 3 2 640 5 1     recv_waking clock 434116 node 0 from 1 type 5 lth 640
-3 -21 0.439659 0 -1 3 2 200 6 1      send entry clock 439658 node 0 to 1 type 6 lth 200
-4 -21 0.440059 0 -1 0                                 send exit clock 440059 node 0
-3 -52 0.440105 0 -1 1 2 -1                        recv_blocking clock 440104 node 0
-4 -21 0.445879 1 -1 0                                 send exit clock 445879 node 1
-3 -52 0.445931 1 -1 1 2 -1                        recv_blocking clock 445930 node 1
-4 -52 0.446151 1 0 3 2 200 6 0     recv_waking clock 446150 node 1 from 0 type 6 lth 200
-3 -901 0.446960 2 -1 0                              trace_start clock 446960 node 2
```

Le logiciel est interactif, l'utilisateur donnant des ordres sous forme d'événements `X-Window` mais ce ne sont pas les seuls événements que connaît le logiciel puisqu'il assure une visualisation animée de l'exécution parallèle et donc est à l'écoute des événements de la trace du programme. Les objectifs principaux mis en avant lors de la conception du logiciel étaient les suivants : facilité de compréhension, simplicité d'utilisation et portabilité.

Puisque l'enjeu de la visualisation est de faciliter la compréhension d'une exécution parallèle, il était impératif que les quelques 25 fenêtres de visualisation soient des plus intuitives. Par ailleurs, pour que le logiciel soit utilisé il devait avant tout être simple et avoir cette possibilité de portage sur de nombreuses plate-formes. Inspiré de divers travaux précédents dans le domaine, `Paragraph` se distingue notamment en offrant de nombreuses vues et en étant apte à visualiser des traces de provenances diverses (le logiciel n'est pas attaché à un type particulier de machines parallèles du moment que le modèle d'exécution est fondé sur des échanges de messages).

`Paragraph` propose à l'utilisateur plusieurs points de vue sur l'exécution de son programme parallèle. Autrement dit, chaque visualisation concerne un point particulier comme les échanges de messages, la nature de ces messages, leur nombre, l'évolution de la charge des machines ou processeurs impliqués et bien d'autres vues qui s'avèrent très utiles. Le graphique ci-contre montre les échanges de messages entre processeurs. C'est la fenêtre `animation` du logiciel qui représente le système parallèle par un graphe dont les sommets sont les processeurs et dont les arêtes montrent les communications entre processeurs. Cet exemple correspond en fait à des processeurs ne communicant qu'au travers de sorte d'anneaux de communication, un anneau regroupant les processeurs de numéros 2 à 7, un anneau étant constitué des processeurs de numéros 8, 10, 11, 12 et 13, un autre liant les processeurs 9, 14, 15, 16 et 17 et enfin un dernier regroupant les processeurs de 18 à 22.





En fait, on peut à l'aide de `Paragraph` mettre en évidence les points faibles d'un programme : cela peut être un trop grand nombre de messages échangés ou une synchronisation trop importante entre les différentes tâches, le tout ralentissant l'exécution du programme. Le logiciel propose par exemple une fenêtre de visualisation du volume d'échange de messages entre les processeurs, illustrée par la figure ci-contre.

Avant de passer au concept de `PGPVM`, il faut remarquer que `Paragraph` visualise plutôt des processeurs que des processus. En effet, l'idée va être dans la suite d'utiliser `Paragraph` pour visualiser des calculs distribués. Plus précisément, `Paragraph` sera utilisé pour visualiser des calculs coupés en tâches qui s'apparentent à des processus physiques se répartissant sur un petit nombre de processeurs.

`Paragraph` peut être considéré comme le logiciel de traçage le plus facile à installer et le plus simple à utiliser. Malgré un nombre confortable de fenêtres de visualisation, la relation entre les données de la trace et le source du programme n'est pas toujours très claire et l'analyse des performances d'une partie précise d'un programme se révèle parfois difficile.

8.5 La surcouche PGPVM

Le lien entre `PVM` et `Paragraph`, c'est Brad Topol qui l'a établi en mettant au point la librairie `PGPVM` dont l'objet est de produire des données de traçage lisibles par `Paragraph` à partir des exécutions de programmes distribués sous environnement `PVM`. Tout programmeur `PVM` peut bien entendu réaliser ce travail lui-même en intégrant aux endroits propices des instructions pour extraire une information concernant par exemple une émission ou réception de message, la création ou terminaison d'une tâche.

Cependant, cela demande un travail non négligeable pour que les informations extraites soient valides ou tout du moins interprétables correctement. Pour illustrer ce point, il faut bien avoir à l'esprit que les différents processus se trouvent sur différentes machines dont les horloges ne sont pas forcément synchronisées et par suite les événements estampillés d'une information de temps n'auront guère de sens si l'on ne reconstruit pas une horloge globale.

Le développement d'outils de visualisation d'exécution concurrente doit suivre des principes précis pour amener un “plus” et ne pas poser plus de problèmes qu'il n'en résout. L'utilisation de l'outil doit demander un effort minimal : cela peut être réalisé en intégrant directement au système de distribution les primitives nécessaires au traçage, *i.e.* en adoptant un mode de traçage automatique qui doit, qui plus est, pouvoir être activé ou non par le programmeur avant chaque exécution, ceci relevant de la souplesse d'utilisation de l'outil, deuxième critère important.

Et puis, si l'on veut que les données de traçage apportent une réelle information, l'outil doit minimiser la perturbation induite par sa présence au cœur de l'exécution. Même si l'on ne peut pas espérer ne pas modifier les performances d'un programme en utilisant un tel outil, son intégration directement au niveau du système est plus satisfaisante qu'une apparition des ordres de traçage au niveau utilisateur. La dernière spécificité de l'outil réside dans sa capacité à créer une horloge logique établissant un certain ordre sur les événements avec un plus s'il est capable d'y associer une horloge réelle amenant ainsi des informations de performance.

PGPVM est finalement un peu à l'image de l'outil qui aurait toutes ces caractéristiques. Surcouche de PVM, il est plus proche du système que de l'utilisateur, ne demande aucun effort pour son utilisateur, est activable par simple recompilation et reconstruit une horloge globale.

L'idée générale du concept PGPVM est une redéfinition des fonctions PVM utiles à une visualisation des performances d'un programme (création de tâches, échanges de messages, etc.) pour y intégrer des ordres explicites de traçage. Ceci est fait grâce au fichier pgpvm.h dont voici un extrait :

```
#ifndef _PGTRACE_H_
#  define _PGTRACE_H_

#  ifndef USE_PGTRACE
   int pg_tids(t,n) int *t;int n; {return (0);}
   int pg_close() {return (0);}
   int pg_outfile(t) char *t; {return (0);}
   int pg_chprefix(t) char *t; {return (0);}
#  else
#    define pvm_mytid pg_mytid
#    define pvm_exit pg_exit
#    define pvm_recv pg_recv
#    define pvm_nrecv pg_nrecv
#    define pvm_trecv pg_trecv
#    define pvm_send pg_send
#    define pvm_mcast pg_mcast
#    define pvm_initsend pg_initsend
#    define pvm_psend pg_psend
#    define pvm_precv pg_precv
     int pg_mytid           __ProtoGlarp__(( void ));
     int pg_exit            __ProtoGlarp__(( void ));
     int pg_recv            __ProtoGlarp__(( int, int ));
     int pg_nrecv           __ProtoGlarp__(( int, int ));
     int pg_trecv           __ProtoGlarp__(( int, int, struct timeval * ));
     int pg_send            __ProtoGlarp__(( int, int ));
     int pg_mcast           __ProtoGlarp__(( int *, int, int ));
     int pg_initsend        __ProtoGlarp__(( int ));
     int pg_psend           __ProtoGlarp__(( int, int, void *, int, int ));
     int pg_precv           __ProtoGlarp__(( int, int, void *, int, int, int *,
                                             int *, int * ));
#  endif
#endif
```

Les fonctions sont redéfinies dans le fichier `pgtracelib.c` qui apporte deux nouvelles fonctions : `pg_tids()` met en route le système de traçage en calculant un temps de référence et `pg_exit()` synchronise par une barrière tous les intervenants afin de faciliter la récupération des données.

Comme nous l'avons dit plus haut, évaluer les performances d'un programme c'est en particulier mesurer l'écoulement du temps. La méthode retenue par Brad Topol fait que chaque tâche calcule son propre temps de référence par rapport auquel les autres temps calculés (envoi et réception de messages) seront comparés. Cela sous-entend, pour que le traçage de l'ensemble soit cohérent, que toutes les tâches administrées exécutent `pg_tids`(1) au même moment. En effet, pour utiliser `PGPVM`, la première tâche lancée doit connaître le nombre exact de processus à lancer, créer ces processus puis appeler `pg_tids`. De leur côté, les processus créés font également `pg_tids` au début de leur exécution. On considère alors que les tâches prennent une référence de temps au même moment.

En conséquence, `PGPVM` ne peut gérer que des exécutions concurrentes dont les tâches naissent toutes ensemble et se terminent en même temps. Pourtant, l'outil `PVM` est plus puissant que cela puisqu'il autorise des tâches à se créer ou à disparaître au cours d'une exécution.

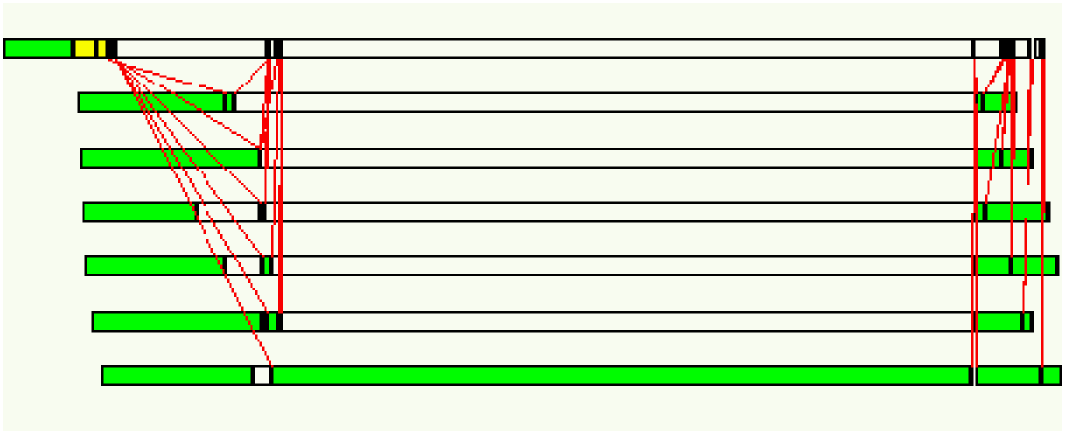


Figure 8.1: *Le processus en haut crée* 6 *fils dont l'un effectue un travail plus long que les* 5 *autres. Chaque fils envoie un message (signifié par un trait sur la figure) au père avant de faire* `pvm_exit` *qui devrait donc le faire disparaître. Nous remarquons ici que tous les fils attendent en fait la fin du dernier processus.*

Lorsque l'on considère des programmes distribués ayant un millier de tâches ou travaux élémentaires à réaliser, on imagine que cela puisse poser problème d'être obligé de maintenir “en vie” un millier de processus du début jusqu'à la fin de l'exécution. Si l'on dispose de 10 machines, cela fait tout de même 100 processus par machine, c'est-à-dire beaucoup trop pour une seule machine.

(1) Le nom de la fonction fait référence à `Paragraph` (`pg`) et aux `tids` de `PVM`. Un `tid` est un identificateur de tâche, équivalent d'un numéro de processus pour une machine parallèle virtuelle.

Dans le modèle de PGPVM, les tâches sont synchronisées malgré elles en fin d'exécution et par conséquent elles se terminent toutes en même temps même si certaines ont depuis longtemps fini leur travail. Une utilisation rationnelle des ressources invite par essence le programmeur à ne pas suivre ce modèle mais de fait, il se prive alors de l'utilisation de PGPVM. Par ailleurs, l'on ne peut plus vraiment dire que le système de traçage minimise la perturbation induite par sa présence comme le montre l'exemple de la figure 8.1 (fenêtre espace/temps de XPVM) qui présente 6 processus attendant (en blanc) une grande partie de leur temps la terminaison du calcul plus long du 7-ième processus.

La modification de la librairie s'imposait car nous ne pouvions même pas l'utiliser pour visualiser les performances de notre programme d'énumération des tableaux de rubans.

8.6 PGPVM nouvelle version

Nous avons ainsi envisagé la modification de PGPVM en collaboration avec Stéphane Bizard, Stephan Dalhem (Université de Marne-la-Vallée) et Brad Topol (*Georgia Institute of Technology*). Nous avons pour cela résolu plusieurs problèmes que nous mentionnons ci-dessous :

① *Qui doit administrer les tâches ?* Nous avons introduit une nouvelle tâche que nous avons appelée administrateur (`pg_administrator`) s'occupant de la gestion complète de la trace d'exécution. Cette tâche connaît l'identité de toutes les tâches tracées, elle fournit le temps de référence à tout le monde, attend les données de traçage, est renseignée sur la terminaison de chacun des intervenants (cela lui permettant de savoir à quel moment elle doit se terminer elle-même) et finalement produit l'ensemble des données collectées dans un unique fichier ;

② *Comment connaître le numéro d'identification de l'administrateur ?* Lorsqu'une tâche effectue un `pvm_spawn` pour créer d'autres tâches, elle transmet les numéros d'identification des nouvelles tâches à l'administrateur et le numéro d'identification de l'administrateur aux nouvelles tâches ;

③ *Quel temps de référence choisir ?* Dans la mesure où nous voulions offrir la possibilité de lancer de nouvelles tâches à n'importe quel moment, il nous fallait un moyen de calculer un temps de référence. Nous avons opter pour la solution visant à établir un temps de référence unique pour toutes les tâches, calculé une seule fois par l'administrateur au début de son exécution. La première fois qu'une tâche exécute un `pg_tids("yes")` pour demander à être administrée, l'administrateur lui transmet le temps de référence ;

④ *Qui lance l'administrateur ?* L'administrateur peut être lancé manuellement ou par exemple en utilisant la nouvelle fonction `pg_startadmin` mise à disposition avec cette nouvelle version de PGPVM. Dans ce cas, c'est la première tâche lancée lors de l'exécution du programme qui inclut cet ordre de création de l'administrateur ;

⑤ *Quel est le nouveau rôle de la fonction `pg_tids` ?* Cette fonction doit être exécutée avant tout autre fonction PVM car elle met la tâche en position de réception bloquante pour recevoir de la part de son père le numéro d'identification de l'administrateur. En conséquence, une tâche ne désirant pas être tracé (tout de suite par exemple) devra quand même effectuer un `pg_tids("N")`—plutôt qu'un `pg_tids("Y")`. Le premier `pg_tids("Y")` effectué par une tâche ouvre un fichier en écriture afin d'y stocker les données de traçage ;

⑥ *A quoi servira la fonction* `pg_exit` *?* Si au moment de l'appel à la fonction `pg_exit`, la tâche est en mode de traçage alors les données stockées seront envoyées à l'administrateur puis la tâche l'informera de sa terminaison. Dans le cas contraire, la tâche informera seulement l'administrateur qu'elle se termine ;

⑦ *Pourra-t-on ne tracer que certaines parties d'un programme ?* La fonction `pg_close` offre cette possibilité très attrayante en stoppant le traçage. On peut d'ailleurs le reprendre en appelant à nouveau `pg_tids("Y")` et ainsi de suite à la guise du programmeur, chaque appel à `pg_close` ayant pour effet d'envoyer les données de traçage à l'administrateur.

Pour bien apprécier les modifications profondes qui ont été effectuées alors que de très légers changements seront visibles du point de vue du programmeur utilisant `PGPVM`, nous remontrons le diagramme espace/temps de la figure 8.1 mais cette fois en utilisant la nouvelle version de `PGPVM`. Le deuxième processus correspond à l'administrateur et l'on voit clairement que les tâches peuvent se terminer lorsqu'elles le désirent :

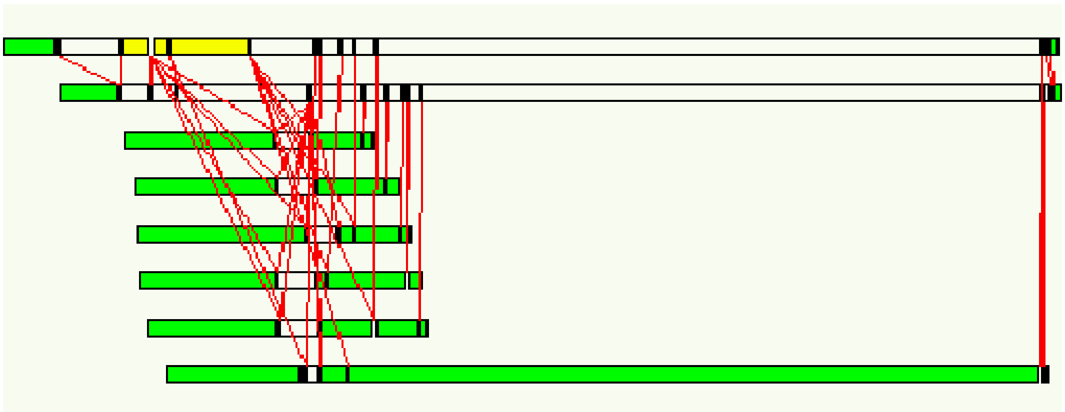


Figure 8.2: *Le premier processus a créé l'administrateur avant de créer ses 6 fils. Les fils n'attendent plus la fin du dernier processus pour se terminer.*

En définitive, les nouvelles fonctionnalités offertes par ce qui s'appelle désormais `PGPVM2` font qu'une tâche peut suspendre son traçage et le reprendre par la suite si elle le désire, qu'une tâche n'est pas obligée de participer au traçage dès sa création et que des tâches peuvent créer à tout moment d'autres tâches qui seront potentiellement tracées. La première version interdit ce schéma classique en demandant une création de toutes les tâches participant au calcul en même temps.

8.7 Application à l'énumération des tableaux de rubans

Nous avons utilisé PGPVM2 pour visualiser les performances de notre programme d'énumération des tableaux de rubans. Cela nous a conduit à apporter une modification à PGPVM2 afin d'intégrer une nouvelle option concernant le lien entre les deux logiciels PGPVM2 et Paragraph. En effet, nous avons vu que le programme ribbon permettait de limiter le nombre maximal de processus esclaves simultanément présents sur la machine virtuelle. L'objet de cette option étant de simuler le comportement du programme sur une machine parallèle réelle ayant un nombre fixe de processeurs, nous avons envisagé la modification de l'administrateur de PGPVM2 afin qu'il prenne en compte ce point de vue.

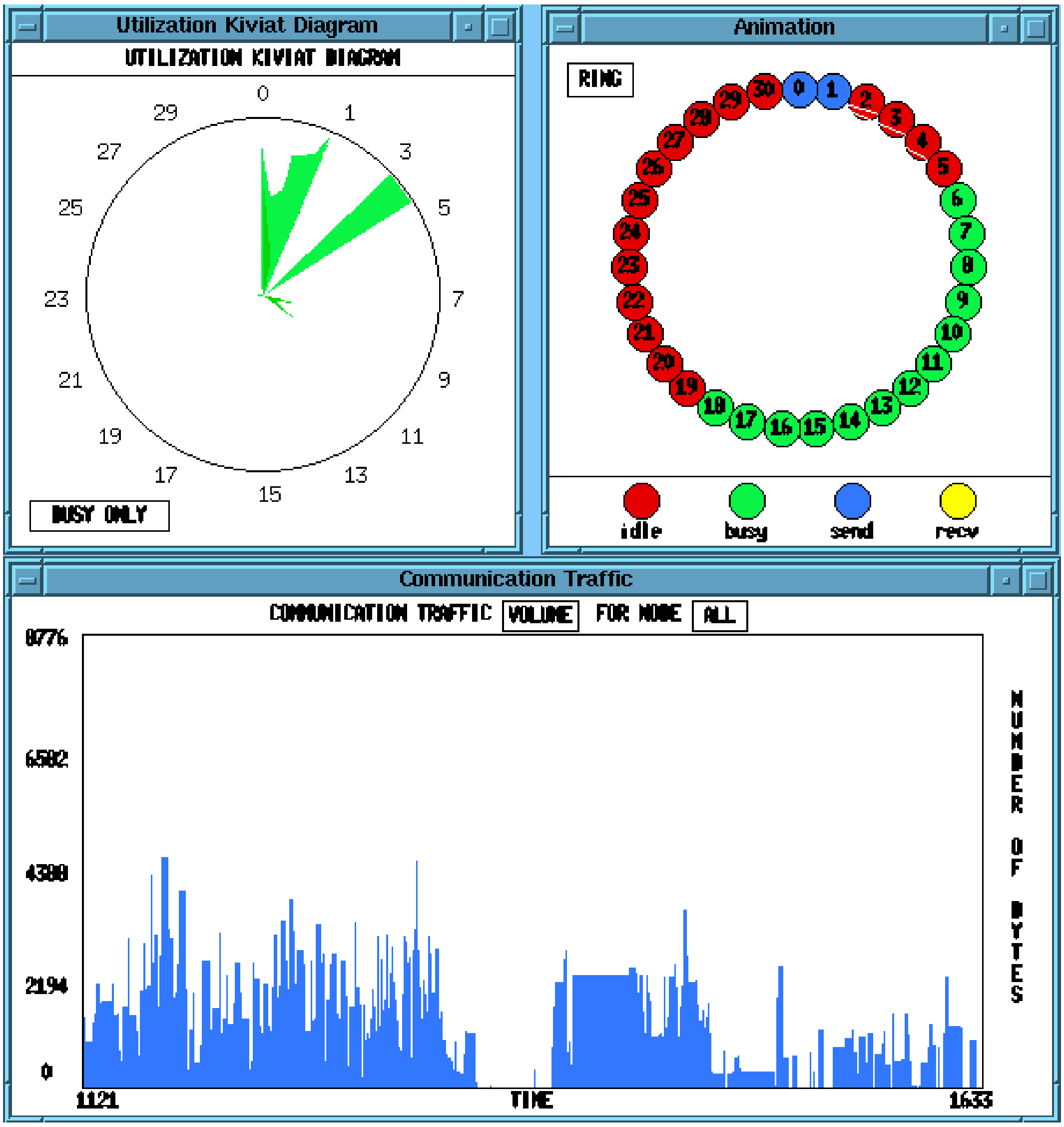
Plus précisément, nous faisions remarquer plus haut que Paragraph visualisait des processeurs et non pas des processus or le fondement de PGPVM repose sur l'identification des deux entités au niveau des données produites. C'est aussi le cas pour PGPVM2 mais avec cette nouvelle option pour le lancement de l'administrateur, nous pouvons fixer le nombre maximal de tâches en concurrence sur la machine virtuelle et ainsi indiquer à l'administrateur qu'il gère un nombre fixe de processus ou plus exactement de processeurs. Par suite, l'identification entre les deux entités est faite au niveau de la gestion des données de traçage.

Dans le cas de l'énumération des tableaux de rubans, Paragraph ne verra que 10 processeurs par exemple alors qu'en fait plus de 1000 processus se seront exécutés pour venir à bout du calcul, 10 tout au plus simultanément.

Nous présentons ci-dessous, à titre indicatif, quelques figures illustrant le déroulement du calcul de l'ensemble des tableaux de rubans de forme $(15, 15, 15, 15, 15, 15, 15, 15)$ et de poids $(4, 4, 4, 4, 4, 4)$. Pour ce calcul, nous avons utilisé une machine virtuelle à base de 8 processeurs et nous avons volontairement fixé à 30 le nombre de processus pouvant exister simultanément :

```
$ timex ribbon -nb -labels -execstats -tasks 30

Execution statistics:
     total user time: 123.71 sec.
      total sys time: 6.53 sec.
    master user time: 74.83 sec.
     master sys time: 3.06 sec.
     tasks user time: 48.88 sec.
      tasks sys time: 3.47 sec.
      total nb tasks: 3947
       total spawned: 75
   max spawned tasks: 30
         max nb tasks: 394

real     4:03.59
user     1:16.79
sys         3.33
```

Les figures suivantes montrent trois fenêtres de visualisation fournies par Paragraph. Le diagramme de Kiviat (en haut à gauche) représente l'ensemble des processeurs autour d'un cercle. Le pourcentage d'utilisation de chaque processeur est représenté par un rayon de cercle. Plus le rayon du cercle est grand, plus le pourcentage est important. La fenêtre d'animation (en haut à droite) nous précise qu'à cet instant précis de l'exécution, la moitié des processus est actif. La dernière fenêtre représente le volume de communication en fonction du temps :



L'ensemble des fenêtres de `Paragraph` permet donc d'apprécier le comportement global ou particulier d'un programme parallèle. Grâce à `PGPVM2`, nous sommes capables de visualiser les performances de programmes distribués en environnement `PVM` mais le point fort reste que nous pouvons fixer au niveau de l'administrateur le nombre de processus simultanés sur la machine virtuelle. Ce faisant, nous pouvons apprécier le comportement qu'aurait notre programme sur une machine parallèle réelle et n'envisager effectivement une exécution sur machine parallèle que lorsque nous aurons jugé que notre programme s'y prête bien.

Références bibliographiques

[1] A. Betten, S. Veigneau, and A. Wasserman. Parallel computation in Combinatorics using PVM. *Electr. J. of Discr. Math. and Theor. Comp. Sci.*, (1996). soumis.

[2] A. Geist, A Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing.* MIT press, (1994).

[3] M.T. Heath and J.E. Finger. *ParaGraph: A Tool for Visualizing Performance of Parallel Programs*, (1996).

[4] B. Topol, V. Sunderam, and A. Alund. *PGPVM Performance Visualization Support for PVM*, (1995).

[5] P.H. Worley. *A New PICL Trace File Format*, (1992).

Conclusion

Oui, la bêtise consiste à conclure.
Nous sommes un fil et nous voulons savoir la trame.
[...] *Quel est l'esprit un peu fort qui ait conclu, à commencer par Homère?*
Contentons-nous du tableau, c'est aussi bon.

— ***Gustave Flaubert***, *4 septembre 1850.*

Après les quelques expériences que nous avons pu mener, la voie de la distribution de calculs nous paraît être très porteuse, non seulement en combinatoire mais plus largement dans le cadre du développement d'un outil de calcul symbolique. Le programme distribuant le calcul de π nous a permis d'apprécier les performances de la machine parallèle virtuelle que nous utilisions. La modification de la bibliothèque de calcul sur entiers longs a mis en avant l'attrait d'une utilisation transparente des ressources de la machine virtuelle : sans aucune modification de son programme, l'utilisateur disposera pour certains calculs de plusieurs machines travaillant en concurrence. Enfin, l'algorithme que nous avons mis au point pour l'énumération des tableaux de rubans nous a permis de calculer de très gros polynômes de spin, en quelques minutes seulement. Cependant, la version distribuée de cet algorithme n'a pas donné de bons résultats. Les tâches distribuées n'étant pas assez conséquentes, les coûts de création, communication et terminaison de la tâche se sont révélés trop importants par rapport au gain en temps escompté.

Les essais que nous avons effectués nous ont donc permis de bien mettre en évidence plusieurs prototypes de programmes distribués et de cerner, dans le contexte très particulier de l'utilisation d'une machine parallèle virtuelle, les types d'algorithmes que nous devrons développer pour utiliser au mieux des ressources éparpillées et hétérogènes. Cette première étude des possibilités de distribution de calculs en combinatoire, menée conjointement avec l'Université de Bayreuth, orientera nos futurs développements.

Un prochain objectif sera l'élaboration d'un système de calcul symbolique permettant de manipuler efficacement les objets combinatoires et algébriques nous intéressant. Ce système sera construit presque exclusivement à partir de briques calculatoires spécialisées déjà existantes. Ces systèmes spécifiques et performants dans leur domaine seront les satellites (distribués sur plusieurs machines) d'un logiciel de calcul formel chargé de transmettre les différentes requêtes et de réceptionner les résultats. De fait, nous disposerons de toutes les facilités offertes par un système de calcul symbolique et de toute la puissance des systèmes spécifiques.

Annexes

A

Développements du q-discriminant

Nous avons abordé à la fin du chapitre 2 le problème du développement du q-discrimant. Grâce à un algorithme du à Scharf, Thibon et Wybourne [1], nous avons obtenu les tables de ces développements sur la base des fonctions de Schur jusqu'à $n = 8$. Rappelons que le q-discriminant est défini par :

$$D_n(q) := \prod_{i,j=1}^{n} (x_i + qx_j) \quad ,$$

sa décomposition sur la base des fonctions de Schur donnant la décomposition en irréductibles des puissances extérieures de la représentation adjointe. Nous présentons ci-dessous ces résultats jusqu'à $n = 6$ en indiquant pour chaque fonction de Schur indicée par la partition λ (dénotée par $\{\lambda\}$) son coefficient polynomial en q dans $D_n(q)$.

q-Discriminant pour $n = 1$

{1}	$(q+1)$

q-Discriminant pour $n = 2$

{3,1}	$q(q+1)^2$
{2,2}	$(q^2-q+1)(q+1)^2$

q-Discriminant pour $n = 3$

{5,3,1}	$q^3(q+1)^3$
{5,2,2}	$q^2(q^2-q+1)(q+1)^3$
{4,4,1}	$q^2(q^2-q+1)(q+1)^3$
{4,3,2}	$q(q^2+1)(q^2-q+1)(q+1)^3$
{3,3,3}	$(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^3$

q-Discriminant pour $n = 4$

{7,5,3,1}	$q^6(q+1)^4$
{7,5,2,2}	$q^5(q^2-q+1)(q+1)^4$
{7,4,4,1}	$q^5(q^2-q+1)(q+1)^4$
{7,4,3,2}	$q^4(q^2+1)(q^2-q+1)(q+1)^4$
{7,3,3,3}	$q^3(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^4$
{6,6,3,1}	$q^5(q^2-q+1)(q+1)^4$
{6,6,2,2}	$q^4(q^2-q+1)^2(q+1)^4$
{6,5,4,1}	$q^4(q^2+1)(q^2-q+1)(q+1)^4$
{6,5,3,2}	$q^3(q^2-q+1)(q^2+1)^2(q+1)^4$
{6,4,4,2}	$q^3(q^2+q+1)(q^2-q+1)^2(q+1)^4$
{6,4,3,3}	$q^2(q^2+q+1)(q^2-q+1)^3(q+1)^4$
{5,5,5,1}	$q^3(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^4$
{5,5,4,2}	$q^2(q^2+q+1)(q^2-q+1)^3(q+1)^4$
{5,5,3,3}	$q^3(q^2-q+1)(q^4+1)(q+1)^4$
{5,4,4,3}	$q(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^4$
{4,4,4,4}	$(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^4$

q-Discriminant pour $n = 5$

{9,7,5,3,1}	$q^{10}(q+1)^5$
{9,7,5,2,2}	$q^9(q^2-q+1)(q+1)^5$
{9,7,4,4,1}	$q^9(q^2-q+1)(q+1)^5$
{9,7,4,3,2}	$q^8(q^2+1)(q^2-q+1)(q+1)^5$
{9,7,3,3,3}	$q^7(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^5$
{9,6,6,3,1}	$q^9(q^2-q+1)(q+1)^5$
{9,6,6,2,2}	$q^8(q^2-q+1)^2(q+1)^5$
{9,6,5,4,1}	$q^8(q^2+1)(q^2-q+1)(q+1)^5$
{9,6,5,3,2}	$q^7(q^2-q+1)(q^2+1)^2(q+1)^5$
{9,6,4,4,2}	$q^7(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{9,6,4,3,3}	$q^6(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{9,5,5,5,1}	$q^7(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^5$

{9,5,5,4,2}	$q^6(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{9,5,5,3,3}	$q^7(q^2-q+1)(q^4+1)(q+1)^5$
{9,5,4,4,3}	$q^5(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{9,4,4,4,4}	$q^4(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^5$
{8,8,5,3,1}	$q^9(q^2-q+1)(q+1)^5$
{8,8,5,2,2}	$q^8(q^2-q+1)^2(q+1)^5$
{8,8,4,4,1}	$q^8(q^2-q+1)^2(q+1)^5$
{8,8,4,3,2}	$q^7(q^2+1)(q^2-q+1)^2(q+1)^5$
{8,8,3,3,3}	$q^6(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^5$
{8,7,6,3,1}	$q^8(q^2+1)(q^2-q+1)(q+1)^5$
{8,7,6,2,2}	$q^7(q^2+1)(q^2-q+1)^2(q+1)^5$
{8,7,5,4,1}	$q^7(q^2-q+1)(q^2+1)^2(q+1)^5$
{8,7,5,3,2}	$q^6(q^2-q+1)(q^2+1)^3(q+1)^5$
{8,7,4,4,2}	$q^6(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{8,7,4,3,3}	$q^5(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{8,6,6,4,1}	$q^7(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{8,6,6,3,2}	$q^6(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{8,6,5,5,1}	$q^6(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{8,6,5,4,2}	$q^5(q^2+q+1)(q^2+1)^2(q^2-q+1)^2(q+1)^5$
{8,6,5,3,3}	$q^5(q^2+q+1)^2(q^2-q+1)^3(q+1)^5$
{8,6,4,4,3}	$q^4(q^2+q+1)^2(q^2-q+1)^4(q+1)^5$
{8,5,5,5,2}	$q^5(q^2+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{8,5,5,4,3}	$q^4(q^4+1)(q^2+1)^2(q^2-q+1)^2(q+1)^5$
{8,5,4,4,4}	$q^3(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{7,7,7,3,1}	$q^7(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^5$
{7,7,7,2,2}	$q^6(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^5$
{7,7,6,4,1}	$q^6(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{7,7,6,3,2}	$q^5(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^5$
{7,7,5,5,1}	$q^7(q^2-q+1)(q^4+1)(q+1)^5$
{7,7,5,4,2}	$q^5(q^2+q+1)^2(q^2-q+1)^3(q+1)^5$

{7,7,5,3,3}	$q^4(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2-q+1)(q+1)^5$
{7,7,4,4,3}	$q^5(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{7,6,6,5,1}	$q^5(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^5$
{7,6,6,4,2}	$q^4(q^2+q+1)^2(q^2-q+1)^4(q+1)^5$
{7,6,6,3,3}	$q^5(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{7,6,5,5,2}	$q^4(q^4+1)(q^2+1)^2(q^2-q+1)^2(q+1)^5$
{7,6,5,4,3}	$q^3(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^5$
{7,6,4,4,4}	$q^4(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2-q+1)^2(q+1)^5$
{7,5,5,5,3}	$q^3(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{7,5,5,4,4}	$q^2(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^5$
{6,6,6,6,1}	$q^4(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^5$
{6,6,6,5,2}	$q^3(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^5$
{6,6,6,4,3}	$q^4(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2-q+1)^2(q+1)^5$
{6,6,5,5,3}	$q^2(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^5$
{6,6,5,4,4}	$q^3(q^4+q^3+q^2+q+1)(q^2-q+1)(q^4-q^3+q^2-q+1)^2(q+1)^5$
{6,5,5,5,4}	$q(q^2-q+1)(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q+1)^5$
{5,5,5,5,5}	$(q^4-q^3+q^2-q+1)(q^6-q^3+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^5$

q-Discriminant pour $\mathrm{n}=6$

{11,9,7,5,3,1}	$q^{15}(q+1)^6$
{11,9,7,5,2,2}	$q^{14}(q^2-q+1)(q+1)^6$
{11,9,7,4,4,1}	$q^{14}(q^2-q+1)(q+1)^6$
{11,9,7,4,3,2}	$q^{13}(q^2-q+1)(q^2+1)(q+1)^6$
{11,9,7,3,3,3}	$q^{12}(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{11,9,6,6,3,1}	$q^{14}(q^2-q+1)(q+1)^6$
{11,9,6,6,2,2}	$q^{13}(q^2-q+1)^2(q+1)^6$
{11,9,6,5,4,1}	$q^{13}(q^2-q+1)(q^2+1)(q+1)^6$
{11,9,6,5,3,2}	$q^{12}(q^2-q+1)(q^2+1)^2(q+1)^6$
{11,9,6,4,4,2}	$q^{12}(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,9,6,4,3,3}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$

{11,9,5,5,5,1}	$q^{12}(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{11,9,5,5,4,2}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{11,9,5,5,3,3}	$q^{12}(q^2-q+1)(q^4+1)(q+1)^6$
{11,9,5,4,4,3}	$q^{10}(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,9,4,4,4,4}	$q^9(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{11,8,8,5,3,1}	$q^{14}(q^2-q+1)(q+1)^6$
{11,8,8,5,2,2}	$q^{13}(q^2-q+1)^2(q+1)^6$
{11,8,8,4,4,1}	$q^{13}(q^2-q+1)^2(q+1)^6$
{11,8,8,4,3,2}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{11,8,8,3,3,3}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{11,8,7,6,3,1}	$q^{13}(q^2-q+1)(q^2+1)(q+1)^6$
{11,8,7,6,2,2}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{11,8,7,5,4,1}	$q^{12}(q^2-q+1)(q^2+1)^2(q+1)^6$
{11,8,7,5,3,2}	$q^{11}(q^2-q+1)(q^2+1)^3(q+1)^6$
{11,8,7,4,4,2}	$q^{11}(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,8,7,4,3,3}	$q^{10}(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{11,8,6,6,4,1}	$q^{12}(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,8,6,6,3,2}	$q^{11}(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,8,6,5,5,1}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{11,8,6,5,4,2}	$q^{10}(q^2+q+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{11,8,6,5,3,3}	$q^{10}(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{11,8,6,4,4,3}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{11,8,5,5,5,2}	$q^{10}(q^2+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,8,5,5,4,3}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{11,8,5,4,4,4}	$q^8(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,7,7,7,3,1}	$q^{12}(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{11,7,7,7,2,2}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{11,7,7,6,4,1}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{11,7,7,6,3,2}	$q^{10}(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{11,7,7,5,5,1}	$q^{12}(q^2-q+1)(q^4+1)(q+1)^6$

{11,7,7,5,4,2}	$q^{10}(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{11,7,7,5,3,3}	$q^9(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2-q+1)(q+1)^6$
{11,7,7,4,4,3}	$q^{10}(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,7,6,6,5,1}	$q^{10}(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{11,7,6,6,4,2}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{11,7,6,6,3,3}	$q^{10}(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,7,6,5,5,2}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{11,7,6,5,4,3}	$q^8(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{11,7,6,4,4,4}	$q^9(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{11,7,5,5,5,3}	$q^8(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,7,5,5,4,4}	$q^7(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{11,6,6,6,6,1}	$q^9(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{11,6,6,6,5,2}	$q^8(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{11,6,6,6,4,3}	$q^9(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{11,6,6,5,5,3}	$q^7(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{11,6,6,5,4,4}	$q^8(q^4+q^3+q^2+q+1)(q^2-q+1)(q^4-q^3+q^2-q+1)^2(q+1)^6$
{11,6,5,5,5,4}	$q^6(q^2-q+1)(q^4-q^3+q^2-q+1)(q^2+1)(q^4+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q+1)^6$
{11,5,5,5,5,5}	$q^5(q^4-q^3+q^2-q+1)(q^6-q^3+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{10,10,7,5,3,1}	$q^{14}(q^2-q+1)(q+1)^6$
{10,10,7,5,2,2}	$q^{13}(q^2-q+1)^2(q+1)^6$
{10,10,7,4,4,1}	$q^{13}(q^2-q+1)^2(q+1)^6$
{10,10,7,4,3,2}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,10,7,3,3,3}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{10,10,6,6,3,1}	$q^{13}(q^2-q+1)^2(q+1)^6$
{10,10,6,6,2,2}	$q^{12}(q^2-q+1)^3(q+1)^6$
{10,10,6,5,4,1}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,10,6,5,3,2}	$q^{11}(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,10,6,4,4,2}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,10,6,4,3,3}	$q^{10}(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{10,10,5,5,5,1}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$

{10,10,5,5,4,2}	$q^{10}(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{10,10,5,5,3,3}	$q^{11}(q^4+1)(q^2-q+1)^2(q+1)^6$
{10,10,5,4,4,3}	$q^9(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,10,4,4,4,4}	$q^8(q^6-q^5+q^4-q^3+q^2-q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{10,9,8,5,3,1}	$q^{13}(q^2-q+1)(q^2+1)(q+1)^6$
{10,9,8,5,2,2}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,9,8,4,4,1}	$q^{12}(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,9,8,4,3,2}	$q^{11}(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,9,8,3,3,3}	$q^{10}(q^4-q^3+q^2-q+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,9,7,6,3,1}	$q^{12}(q^2-q+1)(q^2+1)^2(q+1)^6$
{10,9,7,6,2,2}	$q^{11}(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,9,7,5,4,1}	$q^{11}(q^2-q+1)(q^2+1)^3(q+1)^6$
{10,9,7,5,3,2}	$q^{10}(q^2-q+1)(q^2+1)^4(q+1)^6$
{10,9,7,4,4,2}	$q^{10}(q^2+q+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,9,7,4,3,3}	$q^9(q^2+q+1)(q^2+1)^2(q^2-q+1)^3(q+1)^6$
{10,9,6,6,4,1}	$q^{11}(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{10,9,6,6,3,2}	$q^{10}(q^2+q+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,9,6,5,5,1}	$q^{10}(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,9,6,5,4,2}	$q^9(q^2+q+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{10,9,6,5,3,3}	$q^9(q^2+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{10,9,6,4,4,3}	$q^8(q^2+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{10,9,5,5,5,2}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,9,5,5,4,3}	$q^8(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{10,9,5,4,4,4}	$q^7(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,8,8,6,3,1}	$q^{12}(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{10,8,8,6,2,2}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,8,8,5,4,1}	$q^{11}(q^2+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{10,8,8,5,3,2}	$q^{10}(q^2+q+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,8,8,4,4,2}	$q^{10}(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$

{10,8,8,4,3,3}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{10,8,7,7,3,1}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,8,7,7,2,2}	$q^{10}(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{10,8,7,6,4,1}	$q^{10}(q^2+q+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,8,7,6,3,2}	$q^9(q^2+q+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{10,8,7,5,5,1}	$q^{10}(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{10,8,7,5,4,2}	$q^9(q^2+q+1)(2q^6+4q^4+q^3+4q^2+2)(q^2-q+1)^2(q+1)^6$
{10,8,7,5,3,3}	$q^8(q^6+3q^4+2q^3+3q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,8,7,4,4,3}	$q^8(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{10,8,6,6,5,1}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{10,8,6,6,4,2}	$q^8(q^6+3q^4+2q^3+3q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,8,6,6,3,3}	$q^9(q^2+q+1)(2q^2+3q+2)(q^2-q+1)^4(q+1)^6$
{10,8,6,5,5,2}	$q^8(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{10,8,6,5,4,3}	$q^7(q^2+1)(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{10,8,6,4,4,4}	$q^7(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{10,8,5,5,5,3}	$q^7(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{10,8,5,5,4,4}	$q^6(q^4+1)(q^2+q+1)^2(q^2-q+1)^5(q+1)^6$
{10,7,7,7,4,1}	$q^{10}(q^2+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{10,7,7,7,3,2}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,7,7,6,5,1}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{10,7,7,6,4,2}	$q^8(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{10,7,7,6,3,3}	$q^8(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2-q+1)^2(q+1)^6$
{10,7,7,5,5,2}	$q^8(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2-q+1)^2(q+1)^6$
{10,7,7,5,4,3}	$q^7(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{10,7,7,4,4,4}	$q^8(q^8+q^7+2q^6+q^5+q^4+q^3+2q^2+q+1)(q^2-q+1)^3(q+1)^6$
{10,7,6,6,6,1}	$q^8(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{10,7,6,6,5,2}	$q^7(q^2+q+1)(q^4+1)(q^2+1)^2(q^2-q+1)^3(q+1)^6$
{10,7,6,6,4,3}	$q^7(q^2+1)(q^4+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{10,7,6,5,5,3}	$q^6(q^2+1)(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$

{10,7,6,5,4,4}	$q^6(q^2-q+1)^2(q^4+1)^2(q^2+1)^3(q+1)^6$
{10,7,5,5,5,4}	$q^5(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q^2+1)^2(q^4+1)^2(q+1)^6$
{10,6,6,6,6,2}	$q^7(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{10,6,6,6,5,3}	$q^6(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{10,6,6,6,4,4}	$q^7(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{10,6,6,5,5,4}	$q^5(q^4+q^3+q^2+q+1)(q^2+q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^3(q+1)^6$
{10,6,5,5,5,5}	$q^4(q^6-q^5+q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{9,9,9,5,3,1}	$q^{12}(q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{9,9,9,5,2,2}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{9,9,9,4,4,1}	$q^{11}(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{9,9,9,4,3,2}	$q^{10}(q^4-q^3+q^2-q+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{9,9,9,3,3,3}	$q^9(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{9,9,8,6,3,1}	$q^{11}(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,9,8,6,2,2}	$q^{10}(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{9,9,8,5,4,1}	$q^{10}(q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,9,8,5,3,2}	$q^9(q^2+q+1)(q^2+1)^2(q^2-q+1)^3(q+1)^6$
{9,9,8,4,4,2}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{9,9,8,4,3,3}	$q^8(q^2+q+1)^2(q^2-q+1)^5(q+1)^6$
{9,9,7,7,3,1}	$q^{12}(q^2-q+1)(q^4+1)(q+1)^6$
{9,9,7,7,2,2}	$q^{11}(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,9,7,6,4,1}	$q^{10}(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{9,9,7,6,3,2}	$q^9(q^2+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{9,9,7,5,5,1}	$q^9(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2-q+1)(q+1)^6$
{9,9,7,5,4,2}	$q^8(q^6+3q^4+2q^3+3q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,9,7,5,3,3}	$q^8(q^2-q+1)(q^{12}+2q^{10}+6q^8+5q^6+6q^4+2q^2+1)(q+1)^6$
{9,9,7,4,4,3}	$q^7(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{9,9,6,6,5,1}	$q^{10}(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,9,6,6,4,2}	$q^9(q^2+q+1)(2q^2+3q+2)(q^2-q+1)^4(q+1)^6$
{9,9,6,6,3,3}	$q^8(q^8-2q^7+4q^6-q^5+q^4-q^3+4q^2-2q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{9,9,6,5,5,2}	$q^8(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2-q+1)^2(q+1)^6$

{9,9,6,5,4,3}	$q^7(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{9,9,6,4,4,4}	$q^6(q^{14}-2q^{13}+4q^{12}-4q^{11}+8q^{10}-7q^9+10q^8-7q^7+10q^6-7q^5+8q^4-4q^3+4q^2-2q+1)(q^2-q+1)^2(q+1)^6$
{9,9,5,5,5,3}	$q^8(q^2+q+1)(q^2-q+1)^2(q^4+1)^2(q+1)^6$
{9,9,5,5,4,4}	$q^7(q^2+q+1)(q^4+1)^2(q^2-q+1)^3(q+1)^6$
{9,8,8,7,3,1}	$q^{10}(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{9,8,8,7,2,2}	$q^9(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,8,8,6,4,1}	$q^9(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{9,8,8,6,3,2}	$q^8(q^2+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{9,8,8,5,5,1}	$q^{10}(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,8,8,5,4,2}	$q^8(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{9,8,8,5,3,3}	$q^7(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2+q+1)(q^2-q+1)^2(q+1)^6$
{9,8,8,4,4,3}	$q^8(q^4+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{9,8,7,7,4,1}	$q^9(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{9,8,7,7,3,2}	$q^8(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{9,8,7,6,5,1}	$q^8(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{9,8,7,6,4,2}	$q^7(q^2+1)(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{9,8,7,6,3,3}	$q^7(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{9,8,7,5,5,2}	$q^7(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2+1)(q^2-q+1)^2(q+1)^6$
{9,8,7,5,4,3}	$q^6(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{9,8,7,4,4,4}	$q^7(q^8+q^7+2q^6+q^5+q^4+q^3+2q^2+q+1)(q^2+1)(q^2-q+1)^3(q+1)^6$
{9,8,6,6,6,1}	$q^9(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{9,8,6,6,5,2}	$q^7(q^2+1)(q^4+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{9,8,6,6,4,3}	$q^6(q^{10}+3q^8+q^7+4q^6+q^5+4q^4+q^3+3q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,8,6,5,5,3}	$q^6(q^4+1)(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{9,8,6,5,4,4}	$q^5(q^{16}-q^{15}+4q^{14}-2q^{13}+9q^{12}-3q^{11}+13q^{10}-4q^9+15q^8-4q^7+13q^6-3q^5+9q^4-2q^3+4q^2-q+1)(q^2-q+1)^2(q+1)^6$
{9,8,5,5,5,4}	$q^6(q^2+1)(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,7,7,7,5,1}	$q^8(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,7,7,7,4,2}	$q^7(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$

{9,7,7,7,3,3}	$q^8(q^2+q+1)(q^2-q+1)^2(q^4+1)^2(q+1)^6$
{9,7,7,6,6,1}	$q^7(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{9,7,7,6,5,2}	$q^6(q^2+1)(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{9,7,7,6,4,3}	$q^6(q^4+1)(q^2+q+1)^3(q^2-q+1)^4(q+1)^6$
{9,7,7,5,5,3}	$q^5(q^{10}-2q^9+4q^8-3q^7+6q^6-5q^5+6q^4-3q^3+4q^2-2q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{9,7,7,5,4,4}	$q^6(q^2+q+1)^2(q^4+1)^2(q^2-q+1)^3(q+1)^6$
{9,7,6,6,6,2}	$q^6(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2+q+1)(q^2-q+1)^4(q+1)^6$
{9,7,6,6,5,3}	$q^5(q^{12}+3q^{10}-q^9+3q^8+5q^6+3q^4-q^3+3q^2+1)(q^2+q+1)(q^2-q+1)^3(q+1)^6$
{9,7,6,6,4,4}	$q^5(q^2+q+1)^2(q^4+1)^2(q^2-q+1)^4(q+1)^6$
{9,7,6,5,5,4}	$q^4(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2+q+1)^2(q^2-q+1)^5(q+1)^6$
{9,7,5,5,5,5}	$q^5(q^4-q^3+q^2-q+1)(q^4-q^2+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{9,6,6,6,6,3}	$q^5(q^4+q^3+q^2+q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{9,6,6,6,5,4}	$q^4(q^2+1)(q^4+q^3+q^2+q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{9,6,6,5,5,5}	$q^3(q^4+q^3+q^2+q+1)(q^4+1)(q^2-q+1)^2(q^4-q^3+q^2-q+1)^3(q+1)^6$
{8,8,8,8,3,1}	$q^9(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)(q+1)^6$
{8,8,8,8,2,2}	$q^8(q^6-q^5+q^4-q^3+q^2-q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{8,8,8,7,4,1}	$q^8(q^2+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{8,8,8,7,3,2}	$q^7(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{8,8,8,6,5,1}	$q^9(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{8,8,8,6,4,2}	$q^7(q^4+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{8,8,8,6,3,3}	$q^6(q^{14}-2q^{13}+4q^{12}-4q^{11}+8q^{10}-7q^9+10q^8-7q^7+10q^6-7q^5+8q^4-4q^3+4q^2-2q+1)(q^2-q+1)^2(q+1)^6$
{8,8,8,5,5,2}	$q^8(q^8+q^7+2q^6+q^5+q^4+q^3+2q^2+q+1)(q^2-q+1)^3(q+1)^6$
{8,8,8,5,4,3}	$q^7(q^8+q^7+2q^6+q^5+q^4+q^3+2q^2+q+1)(q^2+1)(q^2-q+1)^3(q+1)^6$
{8,8,8,4,4,4}	$q^8(q^6+q^4+2q^3+q^2+1)(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{8,8,7,7,5,1}	$q^7(q^4-q^3+q^2-q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{8,8,7,7,4,2}	$q^6(q^4+1)(q^2+q+1)^2(q^2-q+1)^5(q+1)^6$
{8,8,7,7,3,3}	$q^7(q^2+q+1)(q^4+1)^2(q^2-q+1)^3(q+1)^6$
{8,8,7,6,6,1}	$q^8(q^4+q^3+q^2+q+1)(q^2-q+1)(q^4-q^3+q^2-q+1)^2(q+1)^6$
{8,8,7,6,5,2}	$q^6(q^2-q+1)^2(q^4+1)^2(q^2+1)^3(q+1)^6$

{8,8,7,6,4,3}	$q^5(q^{16}-q^{15}+4q^{14}-2q^{13}+9q^{12}-3q^{11}+13q^{10}-4q^9+15q^8-4q^7+13q^6-3q^5+9q^4-2q^3+4q^2-q+1)(q^2-q+1)^2(q+1)^6$
{8,8,7,5,5,3}	$q^6(q^2+q+1)^2(q^4+1)^2(q^2-q+1)^3(q+1)^6$
{8,8,7,5,4,4}	$q^6(q^2-q+1)(q^{16}-q^{15}+4q^{14}-2q^{13}+6q^{12}-3q^{11}+9q^{10}-3q^9+9q^8-3q^7+9q^6-3q^5+6q^4-2q^3+4q^2-q+1)(q+1)^6$
{8,8,6,6,6,2}	$q^7(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{8,8,6,6,5,3}	$q^5(q^2+q+1)^2(q^4+1)^2(q^2-q+1)^4(q+1)^6$
{8,8,6,6,4,4}	$q^4(q^4+1)(q^{14}-2q^{13}+4q^{12}-4q^{11}+8q^{10}-7q^9+10q^8-7q^7+10q^6-7q^5+8q^4-4q^3+4q^2-2q+1)(q^2-q+1)^2(q+1)^6$
{8,8,6,5,5,4}	$q^5(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{8,8,5,5,5,5}	$q^7(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4-q^2+1)(q^2-q+1)^2(q+1)^6$
{8,7,7,7,6,1}	$q^6(q^2-q+1)(q^4-q^3+q^2-q+1)(q^2+1)(q^4+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q+1)^6$
{8,7,7,7,5,2}	$q^5(q^4-q^3+q^2-q+1)(q^2-q+1)^2(q^2+1)^2(q^4+1)^2(q+1)^6$
{8,7,7,7,4,3}	$q^6(q^2+1)(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4+1)(q^2-q+1)^2(q+1)^6$
{8,7,7,6,6,2}	$q^5(q^4+q^3+q^2+q+1)(q^2+q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^3(q+1)^6$
{8,7,7,6,5,3}	$q^4(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2+q+1)^2(q^2-q+1)^5(q+1)^6$
{8,7,7,6,4,4}	$q^5(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^2+q+1)(q^4+1)(q^2-q+1)^3(q+1)^6$
{8,7,7,5,5,4}	$q^5(q^4-q^3+q^2-q+1)(q^{14}+3q^{12}+4q^{10}+q^9+5q^8+q^7+5q^6+q^5+4q^4+3q^2+1)(q^2-q+1)(q+1)^6$
{8,7,6,6,6,3}	$q^4(q^2+1)(q^4+q^3+q^2+q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{8,7,6,6,5,4}	$q^3(q^4+q^3+q^2+q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q^2+1)^2(q+1)^6$
{8,7,6,5,5,5}	$q^4(q^4-q^3+q^2-q+1)(q^4-q^2+1)(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{8,6,6,6,6,4}	$q^3(q^2-q+1)(q^4+q^3+q^2+q+1)(q^4+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^4-q^3+q^2-q+1)^2(q+1)^6$
{8,6,6,6,5,5}	$q^2(q^4+q^3+q^2+q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{7,7,7,7,7,1}	$q^5(q^4-q^3+q^2-q+1)(q^6-q^3+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$
{7,7,7,7,6,2}	$q^4(q^6-q^5+q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{7,7,7,7,5,3}	$q^5(q^4-q^3+q^2-q+1)(q^4-q^2+1)(q^2+q+1)^2(q^2-q+1)^4(q+1)^6$
{7,7,7,7,4,4}	$q^7(q^4-q^3+q^2-q+1)(q^4+q^3+q^2+q+1)(q^4-q^2+1)(q^2-q+1)^2(q+1)^6$
{7,7,7,6,6,3}	$q^3(q^4+q^3+q^2+q+1)(q^4+1)(q^2-q+1)^2(q^4-q^3+q^2-q+1)^3(q+1)^6$
{7,7,7,6,5,4}	$q^4(q^4-q^3+q^2-q+1)(q^4-q^2+1)(q^4+1)(q^2-q+1)^2(q^2+1)^3(q+1)^6$
{7,7,7,5,5,5}	$q^5(q^4+q^3+q^2+q+1)(q^4-q^2+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$

{7,7,6,6,6,4}	$q^2(q^4+q^3+q^2+q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^4+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{7,7,6,6,5,5}	$q^3(q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^4-q^2+1)(q^2+q+1)^2(q^2-q+1)^3(q+1)^6$
{7,6,6,6,6,5}	$q(q^4+q^3+q^2+q+1)(q^6-q^3+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^4-q^3+q^2-q+1)^2(q^2-q+1)^2(q+1)^6$
{6,6,6,6,6,6}	$(q^4-q^3+q^2-q+1)(q^6-q^3+1)(q^{10}-q^9+q^8-q^7+q^6-q^5+q^4-q^3+q^2-q+1)(q^6-q^5+q^4-q^3+q^2-q+1)(q^2-q+1)^2(q+1)^6$

Références bibliographiques

[1] T. Scharf, J.-Y. Thibon, and B.G. Wybourne. Powers of the Vandermonde determinant and the quantum Hall effect. *Journal of Physics A*, 27: 4211–4219, (1994).

B

Polynômes de Schubert de B_3

Les polynômes de Schubert [2], définis initialement pour chaque permutation du groupe symétrique, ont trouvé une récente extension au groupe hyperoctaédral. Nous avons vu au chapitre 2 que le groupe hyperoctaédral B_n peut être plongé dans le groupe symétrique $\mathfrak{S}_{2n}$ et que la fonction génératrice des polynômes de Schubert de B_n, introduits par Fomin et Kirillov [1], est donnée par :

$$
\begin{aligned}
\mathrm{FK}(\mathbf{x}) &= B(x_1)\,B(x_2)\ldots B(x_n)\,C'_1(-x_1)\,C'_2(-x_2)\ldots C'_{n-1}(-x_{n-1}) \\
&= \sum_{\nu\in B_n} \mathfrak{X}_\nu(\mathbf{x})\ u_\nu = \sum \mathfrak{X}_\nu(\mathbf{x})\ \partial_\mu \quad ,
\end{aligned}
$$

où les u_ν sont des produits des générateurs $u_0, u_1, \ldots, u_{n-1}$ de l'algèbre nilCoxeter de B_n, où μ est l'image de ν dans $\mathfrak{S}_{2n}$, $B(x) = (1+xu_{n-1})\ldots(1+xu_1)\,(1+xu_0)\,(1+xu_1)\ldots(1+xu_{n-1})$ et $C'_i(x) = (1+xu_{n-1})\,(1+xu_{n-2})\ldots(1+xu_i)$.

Nous donnons ci-dessous la table des $2^3 3!$ polynômes de Schubert de B_3. Nous avons vu au chapitre 2 plusieurs codages des éléments de B_n de sorte que les trois premières colonnes de la table suivante donnent respectivement la permutation signée, la permutation palindrome et le code de chaque élément de B_3. La dernière colonne contient le polynôme de Schubert correspondant :

$\bar{3}1\bar{2}$	536142	033	$x_3^2(x_3+x_2)(x_3+x_1)(x_2+x_1)^2$
$\bar{2}1\bar{3}$	635241	025	$x_3^3(x_3+x_2)(x_3+x_1)(x_2+x_1)^2$
$\overline{123}$	654321	135	$x_3^3x_1x_2^2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{213}$	645231	125	$x_3^3x_1x_2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{312}$	546132	133	$x_3^2x_1x_2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{321}$	456123	123	$x_3x_1x_2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{132}$	564312	134	$x_3^2x_1x_2^2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{231}$	465213	124	$x_3x_1x_2(x_3+x_2)^2(x_3+x_1)(x_2+x_1)$

123	123456	000	1
132	213465	001	$x_1 + x_2 + 2x_3$
$\overline{1}23$	124356	100	$x_3 + x_2 + x_1$
213	132546	010	$x_1 + 2x_2 + 2x_3$
$3\overline{12}$	541632	132	$x_3^2 x_2 (x_3 + x_2)(x_3 + x_1)(x_2 + x_1)$
312	231645	002	$x_1^2 + 2x_1x_2 + 2x_3x_1 + 4x_3x_2 + 2x_3^2 + 2x_2^2$
$\overline{1}32$	214365	101	$(x_1 + x_2 + 2x_3)(x_3 + x_2 + x_1)$
$\overline{2}13$	135246	020	$(x_3 + x_2 + x_1)^2$
$\overline{3}12$	236145	003	$2x_3^2x_1 + 2x_3^2x_2 + x_3^3 + 4x_3x_1x_2 + 2x_3x_2^2 + 2x_3x_1^2 + 2x_1x_2^2 + x_2^3 + 2x_1^2x_2 + x_1^3$
$3\overline{1}2$	241635	102	$(x_3 + x_2)(x_2^2 + 2x_1x_2 + 3x_3x_2 + 2x_3x_1 + x_3^2 + x_1^2)$
$2\overline{1}3$	142536	110	$(x_3 + x_2)(x_3 + x_2 + x_1)$
$1\overline{2}3$	153426	030	$(x_3 + x_2)(x_2^2 + x_3x_2 + x_1x_2 + x_3x_1 + x_3^2)$
$1\overline{3}2$	263415	004	$(x_3 + x_2)(x_2^3 + x_3x_2^2 + x_1x_2^2 + x_3x_1x_2 + x_3^2x_2 + x_3^2x_1 + x_3^3)$
$2\overline{13}$	642531	115	$x_3^3 x_2 (x_3 + x_2)(x_3 + x_1)(x_2 + x_1)$
231	312564	011	$x_1x_2 + 2x_3x_2 + 2x_3x_1 + 2x_3^2$
321	321654	012	$x_1^2x_2 + 2x_1x_2^2 + 6x_3x_1x_2 + 4x_3x_2^2 + 6x_3^2x_2 + 2x_3x_1^2 + 4x_3^2x_1 + 2x_3^3$
$\overline{2}31$	315264	021	$3x_3^2x_2 + 4x_3x_1x_2 + 2x_3x_2^2 + x_1x_2^2 + x_1^2x_2 + 3x_3^2x_1 + 2x_3x_1^2 + x_3^3$
$\overline{3}21$	326154	013	$8x_3^2x_1x_2 + 4x_3^2x_2^2 + 3x_3^3x_2 + 6x_3x_1x_2^2 + 2x_3x_2^3 + 6x_3x_1^2x_2 + x_1x_2^3 + 2x_1^2x_2^2 + x_1^3x_2 + x_3^4 + 3x_3^3x_1 + 4x_3^2x_1^2 + 2x_3x_1^3$
$32\overline{1}$	421653	112	$x_3(x_3 + x_2)(x_3 + x_1)(x_1 + 2x_2 + x_3)$
$23\overline{1}$	412563	111	$x_3(x_3 + x_2)(x_3 + x_1)$
$13\overline{2}$	513462	031	$x_3^2(x_3 + x_2)(x_3 + x_1)$
$12\overline{3}$	623451	005	$(x_3 + x_2)x_3^3(x_3 + x_1)$
$3\overline{21}$	451623	122	$x_3x_2(x_3 + x_2)(x_3 + x_1)(x_2 + x_1)$
$2\overline{31}$	462513	114	$x_3x_2(x_3 + x_2)^2(x_3 + x_1)(x_2 + x_1)$
$\overline{31}2$	246135	103	$(x_3 + x_2)(x_3 + x_1)(x_2 + x_1)(x_3 + x_2 + x_1)$
$\overline{21}3$	145236	120	$(x_3 + x_2)(x_3 + x_1)(x_2 + x_1)$

$\overline{13}2$	264315	104	$(x_3+x_2)(x_2^2+x_3x_2+x_3^2)(x_3+x_1)(x_2+x_1)$
$\overline{12}3$	154326	130	$(x_3+x_2)^2(x_3+x_1)(x_2+x_1)$
$\overline{3}2\overline{1}$	426153	113	$x_3(x_3+x_2)(x_3+x_1)(x_2+x_1)(x_3+x_2+x_1)$
$\overline{2}3\overline{1}$	415263	121	$x_3(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{1}2\overline{3}$	624351	105	$(x_3+x_2)x_3^3(x_3+x_1)(x_2+x_1)$
$\overline{1}3\overline{2}$	514362	131	$x_3^2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$3\overline{2}1$	351624	022	$(x_3+x_2)(x_1x_2^2+2x_3x_2^2+x_1^2x_2+2x_3^2x_2+3x_3x_1x_2+x_3x_1^2+x_3^2x_1)$
$2\overline{3}1$	362514	014	$(x_3+x_2)(x_1x_2^3+2x_3x_2^3+x_1^2x_2^2+2x_3^2x_2^2+3x_3x_1x_2^2+2x_3^3x_2+x_3x_1^2x_2+3x_3^2x_1x_2+x_3^2x_1^2+x_3^3x_1)$
$31\overline{2}$	531642	032	$x_3^2(x_3+x_2)(x_3+x_1)(x_1+2x_2)$
$21\overline{3}$	632541	015	$(x_3+x_2)x_3^3(x_3+x_1)(x_1+2x_2)$
$1\overline{32}$	563412	034	$x_3^2x_2^2(x_3+x_2)(x_3+x_1)(x_2+x_1)$
$\overline{32}1$	356124	023	$(x_3+x_2)(x_3+x_1)(x_2+x_1)(x_3x_1+x_1x_2+x_3x_2)$
$\overline{23}1$	365214	024	$(x_3+x_2)(x_3+x_1)(x_2+x_1)(x_1x_2^2+x_3x_1x_2+x_3^2x_1+x_3^2x_2+x_3x_2^2)$
$1\overline{23}$	653421	035	$x_3^3x_2^2(x_3+x_2)(x_3+x_1)(x_2+x_1)$

Références bibliographiques

[1] S. Fomin and A.N. Kirillov. Combinatorial B_n-analogues of Schubert polynomials, (1994). preprint.

[2] A. Lascoux and M.P. Schützenberger. Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450, (1982).

C
Bibliographie

Car je fais dire aux autres ce que je ne puis si bien dire
tantôt par faiblesse de mon langage,
tantôt par faiblesse de mes sens.
Je ne compte pas mes emprunts, je les pèse.

— ***Montaigne***, *Essais.*

Akl, S.G. (1989). *The design and analysis of parallel algorithms.* Prentice-Hall International Editions.

Ariki, S. (1996).
On the decomposition numbers of the Hecke algebra of $G(m, 1, n)$. preprint.

Banâtre, J.-P. (1991). *La programmation parallèle.* Eyrolles.

Beckmann, P. (1971). *A history of PI.* Barnes & Noble.

Bergeron, F. and Krob, D. (1995).
Acyclic complexes related to noncommutative symmetric functions. preprint, to appear in J. Alg. Comb.

Betten, A., Veigneau, S., and Wasserman, A. (1996).
Parallel computation in Combinatorics using PVM. *Electr. J. of Discr. Math. and Theor. Comp. Sci.* soumis.

Black, G.R.E., King, R.C., and Wybourne, B.G. (1983).
Kronecker products for compact semisimple Lie groups. *J. Phys. A: Math. Gen.*

Carter, R.W. (1986).
Representation theory of the 0-Hecke algebra. *J. of Algebra*, 15: 89–103.

Char, B.W., Geddes, K.O., Gonnet, G.H., Leong, B.L., Monagan, M.B., and Watt, S.M. (1992). *First Leaves: A Tutorial Introduction To Maple V.* Springer-Verlag.

Char, B.W., Geddes, K.O., Gonnet, G.H., Leong, B.L., Monagan, M.B., and Watt, S.M. (1992). *Maple V Language Reference Manual.* Springer-Verlag.

Char, B.W., Geddes, K.O., Gonnet, G.H., Leong, B.L., Monagan, M.B., and Watt, S.M. (1992). *Maple V Library Reference Manual.* Springer-Verlag.

Cherednik, I.V. (1987).
A new interpretation of Gelfand-Tzetlin bases. *Duke. Math. J.*, 54: 563–577.

Cosnard, M. and Trystan, D. (1993). *Algorithmes et architectures parallèles.* InterEditions.

Curtis, C.W. and Reiner, I. (1990). *Methods of representation theory with applications to finite groups and orders*, volume 1. Wiley.

Demazure, M. (1974).
Désingularisation des variétés de Schubert. *Annales E.N.S.*, 6: 163–172.

Dipper, R. and James, G. (1987).
Blocks and idempotents of Hecke algebras of general linear groups. In *Proc. London Math. Soc.*, volume 54, pages 57–82.

Duchamp, G., Krob, D., Leclerc, B., and Thibon, J.-Y. (1996).
Fonctions quasi-symétriques, fonctions symétriques non commutatives, et algèbres de Hecke à $q = 0$. *Comptes Rendus de l'Académie de Sciences*, 322: 107–112.

Fomin, S., Gelfand, S., and Postnikov, A. (1996).
Quantum Schubert polynomials.

Fomin, S. and Greene, C. (1995).
Noncommutative Schur functions and their applications. preprint.

Fomin, S. and Kirillov, A. (1993).
Yang-Baxter equation, symmetric functions and Schubert polynomials. In *Proceedings of the 5-th Conference "Formal Power Series and Algebraic Combinatorics"*, pages 215–229, Florence.

Fomin, S. and Kirillov, A.N. (1994).
Combinatorial B_n-analogues of Schubert polynomials. preprint.

Fulton, W. (1991).
Flags, Schubert polynomials, degeneracy loci and determinantal formulas. *Duke Math. J.*, 65: 381–420.

Fulton, W. and Harris, J. (1991). *Representation theory, a first course.* Springer-Verlag.

Fulton, W. and Lascoux, A. (1994).
A Pieri formula in the Grothendieck ring of a flag bundle. *Duke Mathematical Journal*, 76: 711–729.

Garsia, A.M. and Reutenauer, C. (1989).
A decomposition of Solomon's descent algebra. *Adv. in Math.*, 77: 189–262.

Geist, A., Beguelin, A, Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing.* MIT press.

Gelfand, I.M., Krob, D., Lascoux, A., Leclerc, B., Retakh, V.S., and Thibon, J.-Y. (1995).
Noncommutative symmetric functions. *Advances in Mathematics*, 112: 218–348.

Gelfand, I.M. and Retakh, V.S. (1991).
Determinants of matrices over noncommutative rings. *Funct. Anal. Appl.*, 25: 91–102.

Gelfand, I.M. and Retakh, V.S. (1992).
A theory of noncommutative determinants and characteristic functions of graphs. *Funct. Anal. Appl.*, 26: 1–20. Publ. LACIM, UQAM, Montréal, **14**, 1–26.

Georgi, H. (1982). *Lie algebras in particle physics.* Benjamin, Reading Mass.

Gessel, I. (1984).
Multipartite P-partitions and inner product of skew Schur functions. *Contemporary Math.*, 34: 289–301.

Givental, A. and Kim, B. (1995).
Quantum cohomology of flag manifolds and Toda lattices. *Commun. Math. Phys.*, 168: 609–641.

Grojnowski, I. (1994).
Affine Hecke algebras (and affine quantum GL_n) at roots of unity. *Internat. Res. Math. Notices*, 5: 215–217.

Hayashi, T. (1990).
q-analogues of Clifford and Weyl algebras – spinor and oscillator representations of quantum enveloping algebras. *Commun. Math. Phys.*, 127: 129–144.

Heath, M.T. and Finger, J.E. (1996).
ParaGraph: A Tool for Visualizing Performance of Parallel Programs.

Hirzebruch, F. (1978). *Topological methods in algebraic geometry.* Springer-Verlag. 3rd ed.

Humphreys, J.E. (1990). *Reflection groups and Coxeter groups.* Cambridge University Press, Springer-Verlag.

I.Bernstein, N., I.Gelfand, M., and Gelfand, S.I. (1973).
Schubert cells and cohomology of the spaces G/P. *Uspekhi.Mat.Nauk*, 28: 3–26.

James, G. (1990).
The decomposition matrices of $GL_n(q)$ for $n \leq 10$. *Proc. London Math. Soc.*, 60: 225–265.

James, G. and Kerber, A. (1981). *The representation theory of the symmetric group.* Addison-Wesley.

Kashiwara, M. (1991).
On crystal bases of the q-analogue of universal enveloping algebras. *Duke Math. J.*, 63: 465–516.

Kashiwara, M. (1994).
Crystal bases of modified quantized enveloping algebra. *Duke Math. J.*, 73: 383–413.

Kashiwara, M., Miwa, T., and Stern, E. (1995).
Decomposition of q-deformed Fock spaces. preprint RIMS-1035, Selecta Mathematica (New series), (1996).

Kerber, A. (1991). *Algebraic combinatorics via finite group actions.* Mannheim: BI-Wissenschaftsverlag.

King, R.C. (1971).
Modification rules and products of irreducible representations of the unitary, orthogonal and symplectic groups. *J. Math. Phys.*, 12: 1588–1598.

King, R.C. (1990). *S-functions and characters of Lie algebras and Lie groups*, volume 26 of *Topics in Algebra.* Banach Center Publication, PWN, Warsaw.

Kirillov, A.N., Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1994).
Séries génératrices pour les tableaux de dominos. *Comptes Rendus de l'Académie des Sciences*, 318: 395–400.

Kirillov, A.N. and Maeno, T. (1996).
Quantum Schubert polynomials and the Vafa-Intriligator formula.

Kleshchev, A.S. (1995).
Branching rules for the modular representations of symmetric groups III; some corollaries and a problem of Mullineux. *J. London Math. Soc.*, 2.

Klyachko, A. (1974).
Lie elements in the tensor algebra. *Siberian Math. J.*, 15: 1296–1304.

Knuth, D.E. (1970).
Permutations, matrices and generalized Young tableaux. *Pacific J. Math.*, 34: 709–727.

Knutson, D. (1973).
λ-rings and the representation theory of the symmetric group. *Springer L.N.M.*, 308.

Kohnert, A. (1991).
Weintrauben, Polynome, Tableaux (thèse). *Bayreuther Math. Schr.*, 38: 1–97.

Kohnert, A. and Veigneau, S. (1996).
Using Schubert basis to compute with multivariate polynomials. *Advances in Applied Mathematics.* à paraître.

Kohnert, A. and Veigneau, S. (1996).
Using Schubert toolkit to compute with polynomials in several variables. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 283–293, Minneapolis. University of Minnesota.

Koike, K. and Terada, I. (1987).
Young-diagrammatic methods for the representation theory of the classical groups of type B_n, C_n, D_n. *J. Algebra*, 107: 466–511.

Kostka, C.
Über den Zusammenhang zwischen einingen Formen von symmetrischen Funktionen.

Krob, D., Lascoux, A., Leclerc, B., Thibon, J.-Y., Ung, B.C.V., and Veigneau, S. (1996).
Algebraic Combinatorics with MAPLE and ACE. *Maple Technical Newsletter*. à paraître.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to parallel computing: design and analysis of algorithms*. The Benjamin/Cummings Publishing Company.

Lascoux, A. (1982).
Classes de Chern des variétés de drapeaux. *C.R. Acad. Sci. Paris*, 295: 393–398.

Lascoux, A. (1991).
Cyclic permutations on words, tableaux and harmonic polynomials. In Prakashan, Manoj, editor, *Proc. Hyderabad Conference on Algebraic Groups*, Madras.

Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1993).
Fonctions de Hall-Littlewood et polynômes de Kostka-Foulkes aux racines de l'unité. *Comptes Rendus de l'Académie des Sciences de Paris*, 316: 1–6.

Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1995).
Ribbon tableaux, Hall-Littlewood functions, quantum affine algebras and unipotent varieties. *Journal of Mathematical Physics*. soumis.

Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1996).
Flag varieties and the Yang-Baxter equation. *Letters in Mathematical Physics*. soumis.

Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1996).
Hecke algebras at roots of unity and crystal bases of quantum affine algebras. *Communications in Mathematical Physics*. à paraître.

Lascoux, A., Leclerc, B., and Thibon, J.-Y. (1996).
Twisted action of the symmetric group on the cohomology of a flag manifold. *Banach Center Publications*, 36: 111–124.

Lascoux, A. and Schützenberger, M.P. (1982).
Polynômes de Schubert. *C.R. Acad. Sci. Paris*, 294: 447–450.

Lascoux, A. and Schützenberger, M.P. (1982).
Structure de Hopf de l'anneau de cohomologie et de l'anneau de Grothendieck d'une variété de drapeaux. *C.R. Acad. Sci. Paris*, 295: 629–633.

Lascoux, A. and Schützenberger, M.P. (1983).
Symmetry and flag manifolds. *Springer L.N.*, 996: 118–144.

Lascoux, A. and Schützenberger, M.P. (1985).
Interpolation de Newton à plusieurs variables. *Springer L.N.*, 1146: 161–175.

Lascoux, A. and Schützenberger, M.P. (1985).
Schubert polynomials and the Littlewood-Richardson rule. *Lett. Math. Phys.*, 10: 111–124.

Lascoux, A. and Schützenberger, M.P. (1988).
Arêtes et tableaux. In *Proc. 20-ième séminaire Lotharinghien de Combinatoire*, pages 109–120.

Lascoux, A. and Schützenberger, M.P. (1989).
Fonctorialité des polynômes de Schubert. *Contempory Maths*, 88: 585–598.

Lascoux, A. and Schützenberger, M.P. (1990).
Keys and standard bases, in invariant theory and tableaux.

Lascoux, A. and Schützenberger, M.P. (1992).
Algèbre des différences divisées. *Discrete Math.*, 99: 165–179.

Lascoux, A. and Veigneau, S. (1996).
Double and quantum Pieri formula.

Lewis, T.G. and El-Rewini, H. (1992). *Introduction to parallel computing.* Prentice-Hall International Editions.

Littlewood, D.E. (1950). *The theory of group characters.* Oxford. 2_{nd} ed.

Littlewood, D.E. (1958).
Products and plethysms of characters with orthogonal, symplectic and symmetric groups. *J. Can. Math.*, 10: 17–32.

Littlewood, D.E. (1961).
On certain symmetric functions. *J. London. Math. Soc.*, 11: 485–498.

Littlewood, D.E. and Richardson, A.R.
Group characters and algebra.

Lusztig, G. (1983).
Singularities, character formulas and a q-analog of weight multiplicities. *Analyse et topologie des espaces singuliers (II-III), Astérisque*, 101–102.

Macdonald, I.G. (1991). *Notes on Schubert polynomials*, volume 6. Publ. LACIM, Montréal.

Macdonald, I.G. (1995). *Symmetric functions and Hall polynomials.* Oxford: Clarendon Press. 2_{nd} ed.

MacMahon, P.A. (1915). *Combinatory analysis.* Cambridge University Press. Chelsea reprint, 1960.

Malvenuto, C. (1993).
Produits et coproduits des fonctions quasi-symmétriques et de l'algèbre des descentes (Thèse). Publ. LACIM, UQAM, Montréal.

Malvenuto, C. and Reutenauer, C. (1993).
Duality between quasi-symmetric functions and the solomon descent algebra. preprint UQAM.

Mielnik, B. and Plebański, J. (1970).
Combinatorial approach to Baker-Campbell-Hausdorff exponents. *Ann. Inst. Henri Poincaré*, XII: 215–254.

Misra, K.C. and Miwa, T. (1990).
Crystal base of the basic representation of $U_q(\widehat{sl}_n)$. *Commun. Math. Phys.*, 134: 79–88.

Monk, D. (1959).
The Geometry of flag manifolds. *Proc. London M.S. (3)*, 9: 253–286.

Moody, R.V. and Patera, J.
Fast recursion formula for weight multiplicities.

Pragacz, P. and Ratajski, J.
Formulas for Lagrangian and orthogonal degeneraci loci; the $\widetilde{Q}$-Polynomials Approach. to appear in Compositio Math.

Redfern, D. (1993). *The Maple handbook, Maple V Release 3.* Springer-Verlag.

Reutenauer, C. (1993). *Free Lie algebras.* Oxford University Press.

Sattinger, D.H. and Weaver, O.L. (1986). *Lie groups and algebras with applications to physics, geometry and mechanics*, volume 61. Springer-Verlag, New York.

Scharf, T., Thibon, J.-Y., and Wybourne, B.G. (1994).
Powers of the Vandermonde determinant and the quantum Hall effect. *Journal of Physics A*, 27: 4211–4219.

Schensted, C. (1961).
Longest increasing and decreasing subsequences. *Canad. J. Math.*, 13: 179–191.

Serre, J.-P. (1987). *Complex semisimple Lie algebras.* Springer-Verlag, New York.

Solomon, L. (1976).
A Mackey formula in the group ring of a Coxeter group. *J. of Algebra*, 41: 255–268.

Sottile, F. (1994).
A Geometric Approach to the Combinatorics of Schubert Polynomials.

Stanton, D. (1986). *Invariant theory and tableaux.* Springer-Verlag, New York.

Stanton, D. and White, D. (1985).
A Schensted algorithm for rim-hook tableaux. *J. Comb. Theory A*, 40: 211–247.

Stembridge, J. (1993).
SF, a maple package for symmetric functions. University of Michigan.

Thibon, J.-Y. (1991).
Fonctions symétriques et opérations d'Adams de l'anneau de Grothendieck d'un groupe symétrique. *Comptes Rendus de l'Académie des Sciences*, 313: 829–832.

Thibon, J.-Y. and Veigneau, S. (1996).
Using ACE, an Algebraic Combinatorics Environment for Maple. In *Proc. 8-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 461–468, Minneapolis. University of Minnesota.

Topol, B., Sunderam, V., and Alund, A. (1995).
PGPVM Performance Visualization Support for PVM.

Ung, B.C.V. and Veigneau, S. (1995).
ACE, un environnement de calcul en combinatoire algébrique. In *Proc. 7-th International Conference "Formal Power Series and Algebraic Combinatorics"*, pages 557–562, Noisy-le-Grand. Université de Marne-la-Vallée.

Veigneau, S. (1995).
ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 1.0. Rapport 95–14, IGM.

Veigneau, S. (1996).
ACE, an Algebraic Combinatorics Environment for the computer algebra system MAPLE: User's reference manual, version 2.0. Rapport, IGM.

Veigneau, S. (1996).
Distributed computation of ribbon tableaux and spin polynomials. In Bode, A., Dongara, J., Ludwig, T., and Sunderam, V., editors, *Proc. 3rd European PVM User's Group Meeting*, volume 1156, Munich. Springer-Verlag.

Veigneau, S. (1996).
SP, a package for Schubert polynomials realized with the computer algebra system Maple. *Journal of Symbolic Computation.* à paraître.

Weyl, H. (1939). *Classical groups.* Princeton University Press, Princeton, NJ.

Worley, P.H. (1992).
A New PICL Trace File Format.

Wybourne, B.G.
SCHUR an interactive program for calculating properties of Lie groups.

Wybourne, B.G. (1969). *Symmetry principles and atomic spectroscopy.* Wiley.

D

Index

Les dictionnaires sont les seuls ouvrages dont les mots
sont classés selon un certain ordre—l'alphabétique—qui vaut bien
tous les autres : ordre du jour ou de bataille, ordre des choses ou ordre établi,
ordre des avocats ou ordre dorique,
ordre monastique ou—le pire de tous—l'ordre pour l'ordre.

— ***Georges Elgozy.***

Les mots de cet index sont ordonnés par ordre alphabétique. Les numéros de pages en *italique* font référence à des noms informatiques de programmes, de fonctions ou de bibliothèques de fonctions par exemple. Les numéros de pages figurant en **gras** correspondent à la définition d'un concept.

A

B

C

D

E

P

Q

R

U

V

W

X

Y

Z