Rapport de Stage de Recherche

Une Formalisation des Représentations des Groupes Symétriques en Coq/SSReflect

Thibaut BENJAMIN, X2013
Ecole Polytechnique

LIEU DE STAGE : Laboratoire de Recherche en Informatique (LRI),
Bât 650 Ada Lovelace / Bat 660 Claude Shannon,
Université Paris Sud, Rue Noetzlin,
91190 Gif-sur-Yvette

TUTEUR DE STAGE : Florent HIVERT

DATES DE STAGE : Avril 2016 - Juillet 2016

RAPPORT NON CONFIDENTIEL

Introduction

Le présent rapport a été réalisé dans le cadre d'un stage de recherche effectué LRI (Laboratoire de Recherche en Informatique) sous la direction de Florent Hivert. L'objectif visé pour ce stage est de formaliser des éléments de combinatoires issus de la théorie des représentations des groupes symétriques dans l'outil de preuves assistées par ordinateur Coq/SSReflect.

Le travail au cours de ce stage s'est articulé en trois phases relativement distinctes

— Dans un premier temps, l'enjeu a d'abord été de manipuler suffisamment l'outil de preuve sur des exemples élémentaires, afin d'être ensuite à l'aise pour aborder des exemples plus théoriques.

— Une deuxième partie a été consacrée à la mise en place du cadre théorique des représentations des groupes symétriques, en étudiant les différentes preuves qui ont été données, afin de choisir celle à formaliser

— Enfin la troisième partie, est consacrée à l'intégration de cette théorie en Coq/SSReflect, et à la méthode retenue.

Coq est un assistant de preuves formelles développé par l'INRIA, depuis 1984. A l'origine, il était plutôt développé pour formaliser des preuves de programmes informatiques, mais plusieurs projets ont été menés avec cet outils, dans le but de formaliser des mathématiques. C'est la cas de la bibliothèque Mathematical Components (MathComp), développée entre 2005 et 2013 par Microsoft Research et l'INRIA dans le but de montrer les possibilités de Coq en terme de formalisation mathématique. Cette bibliothèque, que nous allons utiliser, introduit un nouveau langage appelé SSReflect, ainsi que de nombreux résultats d'algèbre, pour une formalisation complète du théorème de Feit-Thomson. L'objectif de ce stage est de s'appuyer sur les résultats déjà présents dans cette bibliothèque pour formaliser la théorie des représentations des groupes symétriques.

Ce rapport suivra le déroulement du stage afin de présenter les résultats obtenus, et la continuation envisagée. On commencera par donner un aperçu du fonctionnement du logiciel Coq, et de sa variante SSReflect. Il s'agit en effet d'un outil très riches, qui repose sur de nombreuses bases théoriques qu'il est important de maîtriser pour comprendre le travail. Des notions de combinatoire élémentaires sur les groupes symétriques seront introduits et formalisés, à titre d'échauffement pour la suite, mais également car ce sont des résultats utiles. Puis quelques éléments de théorie des représentations seront présentés, ils auront pour but d'introduire la théorie qui sera ensuite formalisée en Coq, et ne constituent pas une approche exhaustive. Enfin, la formalisation en Coq des notions présentées sera abordée.

Le travail de programmation réalisé au cours de ce stage est disponible à la page suivante : `https:`

`//github.com/ThiBen/ReprSymGroup` et tous les noms de fichiers et extraits de code fournis dans ce rapport font référence au contenu de cette page, sauf mention explicite du contraire.

Remerciements

Je tiens à remercier Florent Hivert pour l'encadrement qu'il m'a fourni tout au long de ce stage et l'aide qu'il m'a apportée. Je souhaite également remercier, pour les différents conseils qu'ils ont pu apporter, Assia Mahboubi et Georges Gonthier. Je remercie également Vincent Pilaud pour m'avoir permis d'assister au groupe de travail sur la combinatoire, ainsi que pour avoir co-organisé le groupe de travail sur les représentations de groupes. Je remercie aussi les différents membres de l'équipe, stagiaires et doctorants que j'ai cotoyé et qui ont toujours eu la gentillesse de prendre le temps pour m'expliquer leurs sujets d'étude et pour prendre des nouvelles du déroulé de mon stage. Et enfin, je remercie Stéphane Graham-Lengrand pour l'aide qu'il m'a apporté pour trouver ce stage.

Table des matières

Première partie

Le Langage Coq/SSReflect

Chapitre 1

Introduction

La formalisation en Coq/SSReflect est soumise à un certain nombre de contraintes techniques et théoriques liées au système en lui-même et c'est pourquoi nous allons d'abord présenter les enjeux théoriques derrière la preuve formelle, ainsi que les partis pris dans Coq/SSReflect pour répondre à ces enjeux. Pour plus de précisions concernant ces langages, on pourra se référer à la documentation `https://coq.inria.fr/distrib/current/stdlib/` et au manuel de référence en ligne de Coq `https://coq.inria.fr/refman/index.html`, la documentation en ligne de SSReflect `http://math-comp.github.io/math-comp/htmldoc/index.html`, ainsi que le manuel de référence de SSreclect [GMT15] et les notes de cours suivantes [Ser16]

1.1 Le Vocabulaire Utilisé dans Coq

1.1.1 Le Vernaculaire

Le vernaculaire est la sur-couche qui permet à Coq de délimiter ses modes de fonctionnement. En particulier, il délimite le nom, le type et la définition de chaque programme. Coq est doté de deux moyens de définir des programmes, l'un consiste à écrire directement le programme lui-même, le second se fait de manière interactive via l'utilisation du langage des tactiques.

1.1.2 Le Langage Gallina

Le langage de programmation contenu dans Coq, nommé Gallina, qui est essentiellement un langage fonctionnel proche du Caml. Il permet de définir des types inductifs (par exemple, ici une liste d'entiers)

```
Inductive liste : Type :=
 |nil
 |cons: nat -> liste -> liste.
```

ainsi que des fonctions récursives sur ces types inductifs (par exemple, ici la taille d'une liste d'entiers)

```
Fixpoint taille (l:liste) := match l with
  |nil => 0
```

```
  |(cons _ l1)=> 1 + (taille l1)
  end.
```

Remarque. La réponse de Coq à cette dernière commande est alors

```
taille is defined
taille is recursively defined (decreasing on 1st argument)
```

Cela signifie que le système accepte notre définition car il a trouvé un argument strictement décroissant. En effet la spécificité de ce langage est qu'on ne peut définir des fonctions que si l'on prouve leur terminaison en même temps. Cela rend la logique sous-jacente très cohérente, mais signifie aussi qu'il ne peut pas être Turing-complet.

1.1.3 L'Environnement de Preuve et le Langage des Tactiques

Coq est également un outil de preuve qui permet d'énoncer des propositions. Ainsi, par exemple, on peut énoncer la proposition suivante

```
Lemma oddS2 : forall (x : nat), odd x -> odd (x+2).
```

En langage mathématique usuel, cette proposition se lit "pour tout entier x, si x est impair, alors $x+2$ est impair."

Après avoir énoncé une telle proposition, on rentre dans le mode interactif de Coq, dans lequel on va utiliser le langage des tactiques. Le logiciel affiche donc la fenêtre suivante :

```
1 subgoal, subgoal 1 (ID 1)

  ============================
   forall x : nat, odd x -> odd (x + 2)
```

En dessous de la ligne de symboles "=" se trouve ce que l'on cherche à démontrer (but), et au dessus, les hypothèses donc on dispose (contexte). Dans cet exemple le contexte est vide. On va donc maintenant utiliser les tactiques, ce sont des mots-clefs qui permettent de manipuler les éléments du contexte, de les appliquer, de s'en servir pour réécrire ou pour simplifier certains termes... D'une manière générale, les tactiques vont réaliser une modification dans cette fenêtre. Ici, par exemple, après application de tactiques, on obtient :

```
1 subgoal, subgoal 1 (ID 3)

  x : nat
  oddx : odd x
  ============================
   odd (x + 2)
```

1.2 La Bibliothèque MathComp

MathComp a été développée dans le but de se servir de Coq pour formaliser des mathématiques, et il s'agit en réalité de plus qu'une simple bibliothèque puisqu'elle vient avec son propre langage de tactiques, SSReflect.

Nous nous placeront ici dans le cadre de la syntaxe SSReflect uniquement, avec les tactiques qui lui sont propres, sans présenter les tactiques de Coq en général, car ces dernières sont moins adaptées à l'utilisation que l'on souhaite en faire. De plus, SSReflect a été développé avec un grand soin apporté aux conventions de notations, afin que les noms des différents lemmes et théorèmes soient faciles à mémoriser, ce qui n'est pas le cas de la librairie standard dont les notations ne sont pas du tout unifiées, ce qui impose de rechercher beaucoup plus souvent dans la base des lemmes.

Chapitre 2

Correspondance de Curry-Howard et Logique Intuitionniste

2.1 Fonctionnement de Coq

Coq met en oeuvre la correspondance de Curry-Howard, qui établit un isomorphismes entre les règles de typage du λ-calcul et les règles de la logique intuitionniste. Ainsi, il devient équivalent de donner une preuve d'un énoncé, et de construire un λ-terme dont le type est donné. C'est de cette manière que la formalisation se fait, et prouver des énoncés en Coq revient exactement à donner des programmes dont le type est donné.

Une conséquence pratique de ce fonctionnement est, par exemple, qu'au cours d'une preuve le logiciel peut indiquer comme sous-but `nat`, ce qui peut sembler étrange, mais comme pour Coq "démontrer A" est synonyme de "donner un terme dont le type est A", il s'agit en fait juste d'une façon de demander un terme de type `nat`

La Correspondance de Curry-Howard en Action

Nous allons illustrer avec l'exemple de l'implication la manière dont la correspondance de Curry-Howard fonctionne en pratique. Supposons que l'on ait deux propositions A et B et une hypothèse $H : A \implies B$. Si l'on se rappelle que pour le système, `H` est un λ-terme, de type `A -> B`, il est logique de se dire que l'on peut l'évaluer en un argument. On a donc besoin d'un argument `a` de type `A`, c'est à dire en fait une preuve de A, et l'élément `H a` est de type `B` , autrement dit, c'est une preuve B. Ainsi l'hypothèse `H: A->B` est en fait un programme qui transforme toute preuve de A en une preuve de B.

Nous pouvons même observer pousser cet exemple encore plus loin pour comprendre comment fonctionne le langage des tactiques. Il revient en fait à construire un terme dont le type est le type recherché, en indiquant à chaque étape quel est la règle de typage utilisée. Ici, étant donné un but `B`, la tactique `apply H` va remplacer le but `B` par le but `A`. En réalité dans la mémoire, le système est en train de construire un objet de type `B` de la forme `H a` , où `a` est de type `A`. Il ne reste donc plus qu'à construire un objet de type `A`. De la même manière, si l'on dispose d'une hypothèse `H0` de type

`A`, alors la tactique `?applyH in H0` ? va changer le type de `H0` en `B`, c'est à dire qu'il nous signifie qu'il s'ait construire un objet de type `B`, en l'occurence `H H0`

2.2 Conséquences Pratiques

Cette correspondance explique pourquoi les mots-clef `Definition` et `Theorem`, ainsi que toutes leurs variantes sont synonymes. Les objets peuvent être construits explicitement ou implicitement, via le langage des tactiques, dans les deux cas, que ca soit pour définir une fonction ou établir une preuve, la démarche est de construire un objet dont on a annoncé le type. De la même manière les mots-clefs `Axiom`, `Hypothesis` et `Variable` sont synonymes. Il s'agit à chaque fois de choisir un objet générique d'un type donné. L'axiomatisation se fait sur le fait que le type en question est non vide. Cela explique que la syntaxe est la même, la multiplicité des mots-clef ne sert en fait qu'à faciliter la lecture, et à indiquer si le terme qu'on construit doit être retenu en mémoire, ou si il s'agit d'un terme générique et que l'important est que la proposition est vraie.

La conséquence pratique la plus importante de ce fonctionnement est que la correspondance de Curry-Howard impose de raisonner en logique intuitionniste. La grande majorité des mathématiques s'exprime généralement dans le cadre de la logique classique et faire le lien entre les deux est grand enjeu de la formalisation mathématique en Coq. En effet, en toute généralité, il est impossible en Coq d'effectuer un raisonnement par l'absurde, ou de discuter suivant la valeur de vérité d'une proposition. Comme ce sont des raisonnements très utilisés en mathématiques, il est très important de trouver une alternative. Une solution possible est de déclarer comme axiome la proposition $A \vee \neg A$. Cependant, il s'agit là d'un parti pris relativement fort, et il nécessiterait un certain travail afin de rendre cet axiome utilisable en pratique. Dans la suite nous allons présenter l'alternative qui a été optée pour développer la bibliothèque SSReflect.

Chapitre 3

La Réflection Booléenne

3.1 Qu'Est-ce que la Réflection Booléenne ?

3.1.1 Booléens Versus Propositions

Remarquons que l'on dispose en Coq de deux types qui peuvent sembler redondant : les booléens et les propositions. En particulier il existe un `/\` propositionnel et un `&&` booléen, un `\/` propositionnel, et un `||` booléen, et ainsi de suite pour les opérateurs habituels. La différence entre ces deux types est en fait fondamentale, prenons l'exemple de la conjonction pour l'expliquer :

— Dans le cas des propositions, `/\` est un constructeur du type `Prop`, qui est défini inductivement.

— dans le cas des booléens, `&&` est une fonction, c'est à dire un objet de type `bool -> bool -> bool`

La différence entre ces deux visions réside dans le fait que l'on impose dans le langage Coq que les fonctions, par définition, terminent. Ainsi, le type `bool` est construit comme ayant deux valeurs, qui sont `true` et `false`, et possède un certain nombre de fonctions sur lui même, dont la terminaison est prouvée. Par définition, un booléen peut donc s'évaluer, et le résultat sera toujours `true` ou `false`. A l'inverse, le type `Prop` est construit inductivement à partir de deux constantes `True` et `False` et de plusieurs constructeurs de type. Il existe ensuite une procédure de réduction, mais l'arrêt de cette procédure n'est pas jamais garanti, et les propositions permettent donc d'exprimer des énoncés qui sont indécidables, contrairement aux booléens.

3.1.2 Utiliser les Booléens

Remarquons cependant que les booléens permettent naturellement d'exprimer des propositions. Plus précisément, si `A` est une formule booléenne, alors la proposition correspondant `A = true` exprime réellement le même énoncé mathématique. L'idée de la réflection booléenne est d'utiliser cette ambivalence. Ainsi, lorsque c'est possible, on définit une formule booléenne et une proposition pour chacun des énoncés mathématiques que l'on cherche à définir, et on exprime le fait que ces deux formules expriment le même énoncé. On utilise pour cela le mot `reflect` [1]

1. `Inductive reflect (P : Prop) : bool -> Set := ReflectT : P -> reflect P true | ReflectF : P -> reflect P false`

L'avantage de cette démarche est de rétablir en partie la logique classique, du moins pour une certaine classe de propositions. En effet, pour un booléen `b`, on est capable de construire une preuve en Coq de `b = true \/ b = false`. Ainsi, Pour les propositions `P` qui se reflètent sur des booléens, on dispose effectivement de la proposition $P \vee \neg P$ dont la preuve provient de la vision booléenne, et pour ces propositions, la logique intuitionniste coincide avec la logique classique. C'est en fait le cas dès que l'on est capable de prouver qu'une proposition est décidable, mais dans la pratique la seule manière de le faire est d'établir une réflection booléenne.

3.2 Travailler sur un Type Fini

Ce système de réflection booléenne est intéressant dans le cas où l'on travaille sur des types dont l'égalité est décidable, et tout particulièrement lorsque que l'on prête attention aux types finis. Les types avec une égalité décidable, formalisés dans SSReflect sous le nom `eqType` sont intéressant car les propositions de logique non quantifiées traitant de l'égalité peuvent être abordées de la même manière qu'en logique classique. Pour ce qui est des types finis, formalisés sous le nom de `finType`, ce sont déjà des types dont l'égalité est décidable, mais de plus il est tout à fait possible de donner une version booléenne des quantificateurs sur les types finis. En effet, si le type est fini, l'algorithme qui consiste à énumérer tous les éléments du type et à vérifier pour chacun si une propriété donnée est vrai est un algorithme qui termine, et qui peut donc être écrit comme une formule booléenne. Ainsi, sur les types finis, toutes les formules de logique du premier ordre peuvent s'exprimer comme des booléens, et on a une reflection booléenne sur une large classe de proposition. De plus la majorité des constructeurs de type [2] préservent la finitude, ainsi lorsque `A` et `B` sont des types finis, alors `{set A}` et `A-> B` sont encore des types finis, on peut donc quantifier sur ces types et travailler sur la logique d'ordre supérieur de manière classique.

2. sauf constructeur `seq`, qui construit les listes

Chapitre 4

Des Mathématiques en Coq

Le théorème de Feit-Thomson porte sur les groupes finis, et donc permet d'exploiter la réflection booléenne de façon optimale, c'est donc cohérent que ca soit ce théorème qui ait servi d'objectif à la bibliothèque MathComp. Cependant le formalisme développé s'applique également bien à beaucoup de résultats de combinatoire, qui établissent des bijections entre des ensembles finis.

4.1 La Bibliothèque MathComp

Il s'agit de la bibliothèque principale sur laquelle nous allons nous appuyer, et nous détaillons donc son contenu. Il est réparti en plusieurs sous-répertoires :

- ssreflect : Il s'agit du noyau, qui contient le langage de tactiques SSReflect ainsi que les manipulations sur les types primitifs. Ainsi, on y trouve entre autres, la définition de la réflection booléenne, tous les lemmes sur les booléens (`bool`), les entiers (`nat`), les listes sur un type `T` (`seq T`), les types à égalité décidable (`eqType`), les types pour lesquels on dispose d'une procédure de choix (`choiceType`), les types pour lesquels on dispose d'une procédure d'énumération (`finType`), ainsi que les ensembles sur un tel type (`{set T}`).
- fingroup : Il s'agit de la partie contenant des informations sur les groupes finis, en particulier leur définition, les morphismes et automorphismes, groupes quotients les actions de groupe et leurs orbites, et la notion de présentation.
- algebra : C'est la partie contenant tout ce qui est relatif aux anneaux, modules, matrices et polynômes.
- solvable : La partie contenant les propriétés des groupes résolubles.
- field : Le répertoire contenant la définition des corps, et en particulier du corps des complexes algébriques, sur lequel nous nous placeront.
- character : Il s'agit du répertoire contenant une formalisation de la théorie des représentations, et en particulier de la théorie des caractères.

Elle est disponible à l'adresse suivante : `http://math-comp.github.io/math-comp/`

4.2 La Bibliothèque Coq-Combi

Nous utiliseront également la bibliothèque Coq-Combi, développée par Florent Hivert dans le but de formaliser des preuves combinatoires en Coq/SSReflect. Il s'agit s'une formalisation du calcul des coefficients de Littlewood-Richardson sur les fonctions symétriques, et on trouve dans cette bibliothèques les ingrédients combinatoires nécessaires à la formalisation des représentations des groupes symétriques.

Il y a notamment de nombreux résultats sur les partitions d'un entiers, leurs représentations en diagrammes, les tableaux d'Young et leurs manipulations, en particulier l'algorithme d'insertion de Schensted, le tri et les fonctions symétriques. Ce sont les ingrédients de combinatoires que nous allons manipuler dans la suite, en les combinants aux notions de théorie des représentations de MathComp. En particulier, la notion de partition d'un entier nous sera particulièrement utile tout au long de ce rapport, et nous verrons vers la fin des fonctions symétriques, ainsi que des manipulations sur des tableaux, pour établir la correspondance de Robinson-Schensted-Knuth.

Cette bibliothèque se trouve à l'adresse suivante : `https://github.com/hivert/Coq-Combi`

Deuxième partie

Elements de Combinatoire sur les Groupes Symétriques et Leur Formalisation en Coq/SSReflect

Chapitre 5

La Décomposition en Cycle

Nous allons ici établir des résultats préliminaires sur les groupes symétriques qui nous seront indispensables pour la suite, et présenter leur formalisation en Coq/SSReflect. Dans toute la suite, dès que l'on présentera une formalisation, on se limitera aux définitions et aux énoncés des théorèmes clefs pour la démarche. On omettra les définitions et les lemmes plus techniques qui figurent dans le code source du projet, ainsi que les démonstrations. A cet égard, ce rapport peut être utilisé comme un guide de lecture du code. Ce que nous allons présenter dans ce chapitre est le contenu du fichier cycles.v.

Commençons par présenter un théorème bien connu :

Théorème 1 (décomposition en cycle)**.** *Soit $\sigma \in \mathfrak{S}_n$, alors il existe une famille de cycles à supports disjoints $c_1, ..., c_m \in \mathfrak{S}_n$ telle que $\sigma = \prod_{i=1}^{n} c_i$. De plus cette famille est unique à l'ordre près.*

Si l'énoncé de ce théorème peut sembler élémentaire, il faut cependant se rappeler que de nombreuses notions manquent dans la bibliothèque MathComp. En particulier, les notions de cycles et de support d'une permutation ne sont pas implémentées. Il est donc nécessaire de les définir de façon à ce qu'elles soient utilisables et pratiques par la suite.

5.1 Les Notions déjà Présentes dans MathComp

Avant de donner les définitions finalement retenues pour les différentes notions dont nous auront besoin, commençons par nous pencher sur ce qui est déjà présent dans MathComp, et qui pourrait nous être utile. Dans la partie concernant les groupes finis, on trouve une formalisation de la notion d'ensemble de points fixes sous des actions, la notion d'orbite d'une action, ainsi que l'action par permutation. On dispose également de la notion de restriction, qui nous sera utile pour la suite. Nous verrons avec la formalisation comment ces notions peuvent être utilisées en pratique.

5.2 Définition des Notions

Afin de s'appuyer le plus possible sur les notions déjà développées dans la bibliothèque, on définit le support d'une permutation `s` comme étant le complémentaire de l'ensemble des points fixe de l'application.

```
Definition support s := ~:'Fix_('P)([set s])%g.
```

On est ensuite amené à démontrer de nombreux lemmes sur le support, qui ne sont qu'une simple reformulation de ce qu'est le support, mais que nous allons utiliser très souvent. Une autre notion nous sera très utile, il s'agit de la partition du support en orbites non triviales sous l'action de la permutation.

```
Definition psupport s := [set x in pcycles s | #|x| != 1].
```

Cela nous permet très naturellement de définir la notion de cycle, il s'agit simplement d'une permutation qui possède une unique orbite non triviale. Notons que suivant cette convention, l'identité n'est pas un cycle, et la décomposition en cycle de l'identité est donnée par la famille vide, qui est effectivement une famille de cycles à supports disjoints.

```
Definition is_cycle s := #|psupport s| == 1.
```

On peut maintenant donner une construction explicite de la décomposition en cycle, en donnant simplement les restrictions de notre permutation sur chacune de ses orbites non triviales

```
Definition cycle_dec s : {set {perm T}} := [set restr_perm X s | X in psupport s].
```

L'objectif de la suite est de vérifier que cette décomposition est effectivement une décomposition en cycles à support disjoints, et qu'elle est unique.

5.3 La Démonstration Formelle

Pour démontrer le théorème annoncé nous allons tout d'abord montrer que la décomposition que nous avons définie convient bien. Il y a trois conditions à vérifier pour cela :

— Les éléments qui apparaissent dans cette décomposition sont des cycles. Cela est prouvé par le lemme suivant :

```
Lemma is_cycle_dec s : {in (cycle_dec s), forall C, is_cycle C}.
```

— Ils sont à support disjoints. Il est d'abord nécessaire de bien préciser ce que l'on entend par support disjoint

```
Definition disjoint_supports (l: {set {perm T}}):=
  trivIset [set support C| C in l] /\ {in l &, injective support}.
```

Notons que la condition d'injectivité est primordiale car elle permet d'assurer qu'il n'y ait pas dans notre ensemble deux permutations qui ont le même support. On passerai à côté de ce phénomène sans cette condition, car on travail en termes ensemblistes, et les répetitions sont invisibles dans les ensembles. Avec cette définition, on démontre le lemme suivant :

```
Lemma disjoint_cycle_dec s : disjoint_supports (cycle_dec s).
```

— Leur produit est égal à `s` : Cette condition est donnée par le lemme :

```
Lemma cycle_decE s : (\prod_(C in cycle_dec s) C)%g = s.
```

La troisième condition est particulièrement délicate à démontrer, car il s'agit d'évaluer un produit de permutations sur un élément. Cependant on sait que les permutations sont à support disjoint par le point précédent, et cela va beaucoup simplifier la preuve. Nous allons ici utiliser des opérateurs généralisés, appelés bigops, qui possèdent leur propre documentation [BGBP08]. Nous avons prouvé de deux manières la valeur du produit $(\prod_{i=1}^{m} c_i)x$, où les c_i sont des cycles à support disjoints. Le résultat final est donné sous le nom :

```
Lemma prod_of_disjoint (A : {set {perm T}}) C x:
 disjoint_supports A -> C \in A ->
 x \in support C -> (\prod_(C0 in A) C0)%g x = C x.
```

— La première approche consiste à suivre l'élément x à travers chacune des permutations c_i. S'il n'est dans le support d'aucune permutation, alors il est fixé par le produit. Sinon il est dans le support d'exactement l'une d'entre elles, notons i_0 son indice. On a alors

$$\begin{aligned}\left(\prod_{i=1}^{m} c_i\right) x &= \left(\prod_{i=i_0+1}^{m} c_i\right)\left(c_{i_0}\left(\left(\prod_{i=1}^{i_0-1} c_i\right) x\right)\right) \\ &= \left(\prod_{i=i_0+1}^{m} c_i\right)(c_{i_0}(x)) \\ &= c_{i_0}(x)\end{aligned}$$

Cela se justifie par le fait que x n'est dans aucun des supports des $\{c_i\}_{i=1\ldots i_0-1}$, et $c_{i_0}(x)$ n'est dans aucun des supports des $\{c_i\}_{i=i_0+1\ldots m}$, car il est également dans le support de c_{i_0}.

— La seconde approche consiste à intervertir l'ordre des permutations, en remarquant que comme elles sont à support disjoint, elles commutent. Ainsi, si x est dans le support de c_{i_0}, on écrit :

$$\begin{aligned}\left(\prod_{i=1}^{m} c_i\right) x &= c_{i_0}\left(\left(\prod_{i=1, i\neq i_0}^{m} c_i\right) x\right) \\ &= c_{i_0}(x)\end{aligned}$$

C'est la solution qui a été finalement retenue, mais elle est un petit peu plus technique à mettre en oeuvre. En effet, le produit est implémenté sur une loi de groupe, et on peut permuter les élements si cette loi est commutative. Or ici, nous n'avons pas une loi de groupe commutative, mais une famille d'éléments qui commutent. Il s'agit donc de se placer sur le sous groupe engendré par ces éléments pour faire le calcul et vérifier que le produit sur ce sous-groupe coincide bien avec le produit initial.

Les trois critères que nous avons définis peuvent être réunis dans une même formule :

```
Inductive cycle_dec_spec s (A : {set {perm T}}) : Prop :=
  CycleDecSpec of
    {in A, forall C, is_cycle C} &
    disjoint_supports A &
```

```
(\prod_(C in A) C)%g = s : cycle_dec_spec s A.
```

Après avoir montré ces trois critères, démontrer l'unicité de la décomposition en cycles est ne demande pas d'introduire plus de notions. On a donc démontré notre théorème :

```
Theorem cycle_decP s : cycle_dec_spec s (cycle_dec s).
Theorem uniqueness_cycle_dec s (A : {set {perm T}}) : cycle_dec_spec s A -> A = cycle_dec s.
```

Chapitre 6

Type Cyclique et Classes de Conjugaison

Un deuxième résultat que nous allons formaliser avec des techniques élémentaires de combinatoire est la description des classes de conjugaisons de $\mathfrak{S}_n$. Il s'agit ici du résultat formalisé dans le fichier cycletype.v

6.1 Démarche Mathématique

Pour cela, nous allons utiliser la décomposition en cycles précédemment étudiée et définir un invariant que l'ont va appeler type cyclique.

Définition 1. Le type cyclique d'une permutation $\sigma \in \mathfrak{S}_n$ est la liste des longueurs des cycles de la décomposition en cycles de σ, triée par ordre décroissant. C'est une partition de n

Nous allons montrer le théorème suivant

Théorème 2. *Deux permutations sont conjuguées si et seulement si elles ont le même type cycliques. Les classes de conjugaisons de $\mathfrak{S}_n$ sont donc donc en bijection avec les partitions de n*

6.2 Type Cyclique et Conjugaison

Pour formaliser ce résultat, on commence par donner une définition du type cyclique et par le construire en tant que partition.

```
Variable T : finType.
Implicit Types (s t : {perm T}) (n : nat).

Fact cycle_type_partCT (s : {perm T}) :
  is_part_of_n #|T| (parts_shape (pcycles s)).
Definition cycle_type (s : {perm T}) := IntPartN (cycle_type_partCT s).
```

Remarquons que notre définition diffère légèrement, car elle utilise les longueurs des orbites. Il s'agit en fait de la même chose, car les orbites ne sont que les supports des cycles qui apparaissent dans la décomposition en cycles.

On montre ensuite de façon directe que deux permutations conjuguées ont le même type cyclique, simplement en regardant ce que deviennent les orbites sous l'action de la conjugaison.

```
Lemma pcycle_conjg s a x :
  pcycle ((s ^ a)%g) (a x) = [set a y | y in pcycle s x].

Lemma pcycles_conjg s a :
  pcycles (s ^ a)%g = [set [set a y | y in (X : {set T})] | X in pcycles s].

Lemma cycle_type_conjg s a : cycle_type (s ^ a)%g = cycle_type s.
```

Le résultat qui suit est la réciproque, mais elle est exagérément technique. En effet, pour montrer que deux application qui ont le même type cyclique sont conjuguées, il s'agit de construire explicitement une permutation qui les conjugue. Or il existe de nombreuses telles applications. On est donc forcés de faire un choix, concernant la manière d'associer les choses, et aucun choix n'est canonique. En général, si l'ensemble est ordonné, on choisit le plus petit élément, et dans tous les cas, le but est de trouver un critère arbitraire qui décrit l'élément que l'ont va choisir. On ne détaillera pas ici la formalisation, car elle est trop longue et trop technique par rapport à la simplicité du résultat que l'on souhaite. En pratique, si σ et σ' ont le même type cyclique, on ordonne leurs cycles par ordre de longueurs décroissante, on obtient alors les décomposition $\sigma = c_1...c_l$ et $\sigma' = c'_1...c'_l$ puis pour tout $i \leq l$, on choisit une écriture pour les cycles : $c_i = a^i_1...a^i_{k_i}$ et $c'_i = b^i_1...b^i_{k_i}$. Il suffit alors de choisir la permutation qui envoie chacun des a^i_j sur b^i_j, et elle conjugue les deux permutations de départ. Après formalisation de cette construction, on aboutit au théorème suivant.

```
Variable T : finType.
Implicit Types (s t : {perm T}) (C : {set T}) (P : {set {set T}}).

Theorem conj_permP s t :
  reflect (exists c, t = (s ^ c)%g) (cycle_type s == cycle_type t).

Lemma classes_of_permP s t :
  reflect (t \in (s ^: [set: {perm T}])%g) (cycle_type s == cycle_type t).
```

6.3 Les Classes de Conjugaison

Le dernier résultat que nous souhaitons démontrer est que les classes de conjugaisons sont en bijections avec les partitions de n, via le type cyclique. Par le résultat précédent, nous avons déjà montré l'un des sens de la bijection. Pour terminer ce résultat, il s'agit de construire une permutation de type cyclique donné à l'avance, pour montrer que tous les types cycliques sont atteints. Pour cela, on commence par construire explicitement une permutation avec une orbite donnée

```
Implicit Types (l : nat) (ct : intpartn #|T|).
Fact perm_of_pcycle_subproof C : injective (next (enum C)).
Definition perm_of_pcycle C := perm (@perm_of_pcycle_subproof C).
```

Puis on applique cette définition sur un ensemble d'ensembles, en faisant le produit des permutations obtenues individuellement sur chaque ensemble

```
Definition perm_of_parts P :=
  (\prod_(C in [set perm_of_pcycle s | s in [set X in P |#|X|>1]]) C)%g.
```

Pour terminer, choisissons une partition p, il s'agit d'appliquer cette construction à une partition de l'ensemble $\{1...n\}$ dont la suite des longueurs p. On sait qu'une telle partition existe mais on ne l'a pas construite explicitement. On va donc contourner un petit peu cette construction, en montrant l'existence, et en utilisant le fait que l'on est sur un type fini et que grâce à l'énumération, l'existence équivaut à la capacité de le construire explicitement.

```
Lemma perm_of_partCT_exists : exists s, cycle_type s = tc.
Lemma perm_of_partCT_set : {s | cycle_type s == tc}.
Definition perm_of_partCT := val perm_of_partCT_set.
```

Chacune des définitions présentées ici s'accompagne de nombreux lemmes, qui finalement permettent de montrer

```
Lemma perm_of_partCTP : cycle_type perm_of_partCT = tc.
```

On peut maintenant terminer en explicitant la bijection entre les ensembles

```
Definition class_of_partCT := class (perm_of_partCT) [set: {perm T}].


Lemma class_of_partCTP s :
  (cycle_type s == tc) = (s \in class_of_partCT).


Lemma imset_class_of_partCT :
  [set class_of_partCT p | p in setT] = classes [set: {perm T}].
Lemma card_classes_perm :
  #|classes [set: {perm T}]| = #|{: intpartn #|T| }|.
```

6.4 Représentations de $\mathfrak{S}_n$

Jusqu'à maintenant, nous avons formalisé des partitions sur un type fini `T` quelconque, et nous avons donc une partition de l'entier `#|T|`. Nous souhaitons maintenant spécialiser le cas où notre type fini est $\{1, ..., n\}$, noté dans SSReflect `'I_n`. On souhaite en effet obtenir des partitions de `n` et non des partitions de `#|'I_n|`. Les deux nombres sont égaux, mais il est quand même nécessaire de déclarer un cast, comme toujours lorsqu'on a un type dépendant. Ici, on souhaite que cette différence soit implicite, on commence donc par définir une coercion

```
Coercion partCT_of_partn n := intpartn_cast (esym (card_ord n)).
```

On définit ensuite le type cyclique dans le cas d'une permutation sur le type `'I_n`. C'est cette définition que l'on utilisera en pratique plus tard.

```
Variable n : nat.
Definition partn_of_partCT :=
  intpartn_cast (card_ord n) : intpartn #|'I_n| -> intpartn n .
Definition cycle_typeSN (s : 'S_n) : intpartn n :=
  partn_of_partCT (cycle_type s).
```

Troisième partie

Quelques Eléments de Théorie des Représentations

Dans cette partie nous allons présenter des éléments théoriques de la théorie des représentations. Ce sont des notions qui sont déjà formalisées dans la bibliothèque MathComp, qui constituent un socle de connaissances que l'on peut utiliser. Nous énonceront ici les résultats tels qu'ils sont habituellement présentés, et non la manière dont ils sont formalisés, qui peut différer. Cette partie constitue une brève introduction sur le sujet, pour plus de détails, on pourra se référer à [FH91, Ser71]

Dans toute la suite, on se donne un corps k que l'on supposera toujours algébriquement clos et de caractéristique 0. En pratique, on se place très souvent sur $\mathbb{C}$ pour raisonner intuitivement, mais l'implémentation dans Coq/SSReflect nous poussera plutôt à choisir le corps des nombres complexes algébriques.

Chapitre 7

Représentations de Groupes et Représentations d'Algèbre

7.1 Premières Définitions

Dans un premier temps, nous allons définir la notion de représentation de groupe.

7.1.1 Représentations de Groupes

Définition des Représentations

Définition 2 (Représentation d'un groupe)**.** Soit G un groupe fini, une représentation de G sur k est la donnée d'un couple (μ, V_μ), où V_μ est un k-espace vectoriel de dimension finie, et $\mu : G \longrightarrow \mathrm{Aut}(V_\mu)$ est un morphisme de groupes.

La Représentation Triviale

Définition 3. Pour tout groupe G, on a une représentation, que l'on appelle triviale sur G, qui est donnée par (f, k), avec pour tout $g \in G$, $f(g) = 1$. On note cette représentation triv_G.

L'Algèbre de Groupe

Définition 4. Soit $G = \{g_1, ..., g_n\}$ un groupe fini, on appelle l'algèbre du groupe G sur k, notée $k[G]$, l'algèbre sur k dont une base est G et dont la multiplication d'algèbre est donnée sur cette base par le produit dans G.

Proposition 1. *Une k-algèbre de dimension finie est une algèbre de groupe si et seulement si elle possède une base qui est un groupe*

Remarque. Dans le cas échéant, une telle base n'est pas nécessairement unique et une algèbre peut être l'algèbre de deux groupes non isomorphes.

Nous allons maintenant donner une description un peu plus explicite de l'algèbre de groupe.

Proposition 2. *L'espace $k[G]$ est en fait isomorphe, en tant qu'espace vectoriel, à l'espace k^G*

Démonstration. Il suffit d'exhiber deux isomorphismes réciproques, or on a les applications suivantes :

$$\begin{array}{rccc} \Phi : & k^G & \longrightarrow & k[G] \\ & f & \longmapsto & \sum\limits_{g\in G} f(g)g \end{array} \qquad \begin{array}{rccc} \Psi : & k[G] & \longrightarrow & k^G \\ & a=\sum\limits_{g\in G} a_g g & \longmapsto & (g\mapsto a_g) \end{array}$$

On vérifie alors sans difficulté que ce sont deux morphismes et qu'ils sont bien réciproques l'un de l'autre. □

Remarque. $k[G]$ et k^G sont tous deux des k-algèbres, mais elles ne sont pas isomorphes. Cependant on peut définir un produit sur k^G pour que l'isomorphisme précédent soit un isomorphisme d'algèbre.

Définition 5 (produit de convolution sur k^G)**.** On pose $* : k^G \times k^G \longrightarrow k^G$ défini par $\forall f_1, f_2 \in k^G, f_1 * f_2 : g \mapsto \sum\limits_{h\in G} f_1(gh^{-1})f_2(h)$

Proposition 3. *$(k^G, *)$ est une algèbre, et l'isomorphisme entre $k[G]$ et k^G est un isomorphisme d'algèbre pour le produit $*$ de k^G*

La Représentation Régulière

Définition 6 (Représentation Régulière)**.** Soit G un groupe, alors $k[G]$ est une représentation de G pour le morphisme $\mu : G \longrightarrow k[G]$ défini par $\mu(g)(\sum\limits_{h\in G} a_h h) = \sum\limits_{h\in G} a_h(gh)$

On appelle cette représentation la représentation régulière de G, notée Reg_G

Définition 7. La dimension d'une représentation (μ, V_μ) est la dimension de l'espace V_μ en tant qu'espace vectoriel

Remarque. Il arrivera très souvent que l'on note une représentation en donnant simplement l'espace V_μ, ou le morphisme μ, l'autre élément du couple sera alors sous-entendu.

7.1.2 Un Exemple de Représentation

Nous allons ici construire une représentation, qui nous servira d'exemple pour illustrer les différentes notions relatives aux représentations que nous allons définir. On se place sur l'espace vectoriel $\mathbb{C}^6$, et on considère le morphisme ν défini par

$$\forall\sigma\in\mathfrak{S}_3, \forall(c_1,c_2,c_3,c_4,c_5,c_6)\in\mathbb{C}^6, [\nu(\sigma)](c_1,c_2,c_3,c_4,c_5,c_6) = (c_{\sigma(1)}, c_{\sigma(2)}, c_{\sigma(3)}, c_{\sigma(1)+3}, c_{\sigma(2)+3}, c_{\sigma(3)+3})$$

On obtient de cette une représentation du groupe $\mathfrak{S}_3$ sur $\mathbb{C}^6$. On va donner une expression matricielle de cette représentation.

Rappelons qu'une permutation peut s'écrire comme une matrice de 0 et 1, dans laquelle chaque colonne et chaque ligne contient exactement une fois le nombre 1. On lit alors la permutation de la manière suivante : si la case (i, j) contient le nombre 1, alors la permutation envoie le nombre i sur le nombre j. Etant donné une permutation σ, on note alors M_σ la matrice correspondante.

Alors on a le morphisme de groupe suivant, qui définit la représentation :

$$\begin{array}{rccc} \nu : & \mathfrak{S}_3 & \longrightarrow & \mathrm{GL}_6(\mathbb{C}) \simeq \mathrm{Aut}\,\mathbb{C}^6 \\ & \sigma & \longmapsto & \begin{pmatrix} M_\sigma & 0 \\ 0 & M_\sigma \end{pmatrix} \end{array}$$

Dans la suite, la lettre ν sera réservée à cette représentation.

7.2 Réductions et Représentations Irréductibles

7.2.1 Irréductibilité

Définition 8 (sous-représentation)**.** Soit (μ, V_μ) une représentation de G, une sous-représentation de V_μ est un sous-espace vectoriel W de V_μ qui est stable par tous les $\mu(g), g \in G$

Exemple.

- $\{0\}$ et V_μ sont toujours deux sous-représentations de V_μ.
- Reprenons notre représentation ν, et remarquons que le sous espace vectoriel $\mathbb{C}^3 \times \{0\}^3$ de $\mathbb{C}^6$ est stable sous l'action de tous les éléments de $\mathfrak{S}_3$. Ainsi il s'agit d'une sous-représentation de ν

Définition 9 (représentation irréductible)**.** On dit qu'une représentation (μ, V_μ) de A est irréductible si ses seules sous-représentations sont $\{0\}$ et V_μ

Exemple. Une représentation de dimension 1 est toujours irréductible, en effet, si V est une représentation de dimension 1, une sous-représentation W est en particulier un sous-espace vectoriel de V, donc il est de dimension 0 ou 1, c'est à dire, $W = \{0\}$ ou $W = V$

7.2.2 Semi-simplicité

Définition 10 (somme directe de représentation)**.** Etant donné deux représentations (μ, V_μ) et (λ, V_λ), on définit la somme directe de ces deux représentations comme la représentation sur l'espace $V_\mu \oplus V_\lambda$ avec le morphisme défini par

$$V_\mu \oplus V_\lambda \ni x_\mu + x_\lambda \longmapsto \mu(x_\mu) + \lambda(x_\lambda)$$

Théorème 3 (de Maschke)**.** *Pour toute représentation V d'un groupe G, il existe une famille de représentations irréductibles V_ρ tels que $V = \bigoplus V_\rho$, ou de façon équivalente, toute sous-représentation de V admet un supplémentaire qui est stable par l'action de G*

Démonstration. Nous allons prouver l'équivalence de ces deux propriétés par récurrence sur la dimension.

- Pour des représentations de dimension 1, elles sont déjà irréductible, donc ne possèdent pas de sous représentations, et s'écrivent comme somme directe d'une représentation irréductible.
- Soit $n \in \mathbb{N}$, supposons que ces notions soient équivalentes pour les représentations de dimension $k \leq n$. Choisissons alors une représentation V de dimension $n+1$.

— Supposons donc que cette représentation n'est pas irréductible et que toutes ses sous-représentations admettent un supplémentaire stable. Choisissons alors une sous-représentation W qui est donc de dimension $k \leq n$, alors W admet un supplémentaire W' qui est également une sous-représentation, et $\dim W' = k' \leq n$. De plus, dans W et W', toute sous-représentation admet un supplémentaire stable, on peut donc les écrire comme somme d'irréductibles $W = \bigoplus_{i=1}^{m} W_i$ et $W' = \bigoplus_{j=1}^{m'} W'_j$. On obtient ainsi une décomposition en représentations irréductibles de V sous la forme

$$V = \bigoplus_{i=1}^{m} W_i \oplus \bigoplus_{j=1}^{m'} W'_j$$

— Réciproquement, si $V = \bigoplus_{i=1}^{m} V_i$, avec les V_i irréductibles, alors une sous-représentation de V s'écrit $W = \bigoplus_{i \in J} V_i$, avec $J \subset \{1, ..., n\}$. Alors $\bigoplus_{i \in \{1,...,n\} \setminus J} V_i$ est un supplémentaire stable de W.

On admettra que cette propriété est vraie. □

Exemple. Nous allons calculer la décomposition de ν en somme directe de représentations irréductibles.

Remarquons dans un premier temps que

$$\nu = (\mathbb{C}^3 \times \{0\}) \oplus (\{0\} \times \mathbb{C}^3)$$

où l'action de $\mathfrak{S}_3$ sur chacune des composantes est donnée par permutation des composantes.

Sur $\mathbb{C}^3 \times \{0\}$, notons $e_1 = (1, 1, 1, 0, 0, 0)$. On remarque qu'alors $k \cdot e_1$ est stable sous l'action de $\mathfrak{S}_3$, c'est donc une sous représentation, et comme elle est de dimension 1, on en déduit qu'elle est irréductible. L'action de $\mathfrak{S}_3$ sur $k \cdot e_1$ est triviale. On peut également noter qu'en notant $e_2 = (1, -1, 0, 0, 0, 0)$ et $e_3 = (1, 0, -1, 0, 0, 0)$, alors $\mathrm{Vect}(e1, e2)$ est le plan d'équation $c_1 + c_2 + c_3 = 0$ qui est donc stable sous l'action de $\mathfrak{S}_3$. De plus, $\mathrm{Vect}(e1, e2)$ est supplémentaire de $k \cdot e_1$, et on peut vérifier qu'il ne contient pas de sous-représentation stricte (car alors cette représentation serait nécessairement une droite, et aucune droite de ce plan n'est stable).

En procédant de même pour le sous-espace $\{0\} \times \mathbb{C}^3$, on note $e_4 = (0, 0, 0, 1, 1, 1)$, $e_5 = (0, 0, 0, 1, -1, 0)$ et $e_6 = (0, 0, 0, 1, 0, -1)$. On obtient alors la décomposition en représentations irréductibles de la représentation ν

$$\nu = \mathrm{Vect}(e_1) \oplus \mathrm{Vect}(e_2, e_3) \oplus \mathrm{Vect}(e_4) \oplus \mathrm{Vect}(e_5, e_6)$$

7.3 Morphismes de Représentation et Lemme de Schur

7.3.1 Lemme de Schur

Définition 11 (morphisme de représentations)**.** Soient deux représentations (μ, V_μ) et (λ, V_λ). On dit que f est un morphisme de représentations si $f : V_\mu \longrightarrow V_\lambda$ est une application linéaire, et que de plus,

$$\forall g \in G, f \circ \mu(g) = \lambda(g) \circ f$$

On note $\mathrm{Hom}_G(V_\mu, V_\lambda)$ l'ensemble des morphismes de représentation de V_μ dans V_λ

Définition 12 (isomorphisme de représentations)**.** Un morphisme de représentations bijectif est appelé un isomorphisme de représentations. On dit que deux représentations sont isomorphes s'il existe un isomorphisme de représentations de l'une à l'autre

Remarque. Par la suite, nous considèreront les représentations à isomorphisme près, et on simplifiera donc les notations. De plus on note $\widehat{G}$ l'ensemble des représentations irréductibles de G à isomorphisme près.

Exemple. La décomposition de la représentation ν peut donc se réécrire de la manière suivante :

$$\nu = \mathrm{triv}^{\oplus 2} \oplus \rho_2^{\oplus 2}$$

où triv désigne la représentation triviale de $\mathfrak{S}_3$ et ρ_2 désigne la représentation irréductible de dimension 2, $\mathrm{Vect}((1,-1,0),(1,0,-1))$ où l'action de $\mathfrak{S}_3$ est donnée par permutation des composantes

L'ensemble $\mathrm{Hom}_G(V, W)$ va donner de nombreuses informations sur la représentation que l'on étudie, via le lemme suivant :

Théorème 4 (Lemme de Schur)**.** *Soient μ, λ deux représentations irréductibles, alors*

- *si μ et λ ne sont pas isomorphes,* $\mathrm{Hom}_G(V_\mu, V_\lambda) = \{0\}$
- $\mathrm{Hom}_G(V_\mu, V_\mu) = \{c.\,\mathrm{Id}_{V_\mu}, c \in k\}$*, en particulier* $\dim_k \mathrm{Hom}_G(V_\mu, V_\mu) = 1$

7.3.2 Décomposition et Composantes Isotypiques

Composantes Isotypiques et Multiplicités

Définition 13 (Composantes Isotypiques et Multiplicités)**.** Soit (μ, V) une représentation de G, par le théorème de Maschke, il existe une famille de représentations irréductibles V_ρ et des entiers m_ρ tels que $V = \bigoplus V_\rho^{\oplus m_\rho}$

L'espace $V_\rho^{\oplus m_\rho}$ est alors appelé la composante isotypique de ρ dans μ et m_ρ est appelé la multiplicité de ρ dans μ.

Remarque. La décomposition en composantes isotypiques de V est unique, et justifie que l'on ne s'intéresse qu'aux représentations irréductibles

Lemme de Schur et Multiplicités

Dans cette section, nous nous intéresseront aux représentations complètement réductibles, et nous allons appliquer le lemme de Schur à plusieurs reprises pour donner une caractérisation des composantes isotypiques.

Commençons donc par un lemme calculatoire :

Lemme. *L'opérateur* Hom *est compatible aux sommes directes à gauche et à droite, c'est à dire*

- *Soient trois représentations* V_1, V_2, W *alors* $\mathrm{Hom}_G(V_1 \oplus V_2, W) = \mathrm{Hom}_G(V_1, W) \oplus \mathrm{Hom}_G(V_2, W)$
- *Soient trois représentations* V, W_1, W_2 *alors* $\mathrm{Hom}_G(V, W_1 \oplus W_2) = \mathrm{Hom}_G(V, W_1) \oplus \mathrm{Hom}_G(V, W_2)$

On se fixe maintenant une représentation μ quelconque, et pour une représentation irréductible ρ, on note m_ρ la multiplicité de ρ dans μ. On peut alors calculer m_ρ de la manière suivante :

Théorème 5. $m_\rho = \dim \mathrm{Hom}_G(V_\rho, V_\mu)$

Démonstration. En utilisant le lemme de Schur :

$$\begin{aligned}
\mathrm{Hom}_G(V_\rho, \mu) &= \mathrm{Hom}_G(V_\rho, \bigoplus_{\sigma \in \widehat{G}} V_\sigma^{\oplus m_\sigma}) \\
&= \bigoplus_{\sigma \in \widehat{G}} \mathrm{Hom}_G(V_\rho, V_\sigma)^{\oplus m_\sigma} \\
&= \mathrm{Hom}_G(V_\rho, V_\rho)^{\oplus m_\rho} \qquad \text{car } \mathrm{Hom}_G(V_\rho, V_\sigma) = 0 \text{ si } \rho \text{ et } \sigma \text{ ne sont pas isomorphes} \\
&= k^{m_\rho}
\end{aligned}$$

□

Ce théorème montre que calculer les composantes isotypiques de μ revient à calculer les espaces $\mathrm{Hom}_G(\rho, \mu)$

Chapitre 8

Théorie des Caractères et Fonctions Centrales

8.1 Fonctions Centrales

Définition 14 (fonction centrale)**.** On dit qu'une fonction $f : G \to k$ est centrale si $\forall g, h \in G, f(gh) = f(hg)$ ou de façon équivalente $\forall g, h \in G, f(h^{-1}gh) = f(g)$

Remarque. La donnée d'une fonction centrale est strictement équivalente à la donnée d'une fonction $\mathrm{Conj}(G) \to k$. C'est pourquoi ces fonctions sont parfois appelées fonctions de classes. Par la suite, si f est une fonction de classe, et C une classe de conjugaison dans G, on notera $f(C)$ la valeur que prend f sur n'importe quel élément de C.

L'ensemble des fonctions centrales sera noté $\mathcal{CF}_k(G)$, ou simplement $\mathcal{CF}(G)$. Nous allons maintenant considérer le cas des représentations sur $\mathbb{C}$, et étudier la structure de l'ensemble des fonctions centrales.

Proposition 4. *L'ensemble $\mathcal{CF}(G)$ est muni canoniquement d'une structure d'espace vectoriel.*

Définition 15 (produit scalaire)**.** Soient $\phi, \psi : G \longrightarrow k$ deux fonctions centrales, on définit leur produit scalaire $(\phi|\psi) = \frac{1}{|G|} \sum_{g \in G} \phi(g)\psi(g^{-1}) = \frac{1}{|G|} \sum_{C \in \mathrm{Conj}(G)} |C|\phi(C)\psi(C^{-1})$. Ainsi, $\mathcal{CF}(G)$ est muni d'une structure d'espace hermitien.

Définition 16 (Fonction Indicatrice d'une Classe)**.** Soit C une classe de conjugaison dans G, on définit la fonction indicatrice de C, que l'on note par $\mathbb{1}_C$ par $\forall x \in G, \mathbb{1}_C(x) = \delta_{x \in C}$

Proposition 5. *La famille $(\mathbb{1}_C)_{C \in \mathrm{Conj}(G)}$ est une base orthogonale de $\mathcal{CF}(G)$*

8.2 Théorie des Caractères

8.2.1 Définitions et Premières Propriétés

Définition 17 (caractère)**.** Soit μ une représentation d'un groupe G. Son caractère, noté χ_μ est la fonction

$$\begin{array}{rccc} \chi_\mu : & G & \longrightarrow & k \\ & g & \longmapsto & \mathrm{Tr}(\mu(g)) \end{array}$$

Propiétés.

- $\chi_\mu(1) = \dim V_\mu$
- χ_μ *est une fonction centrale*
- *si* μ *et* μ' *sont isomorphes,* $\chi_\mu = \chi_{\mu'}$
- $\chi_{\mu\oplus\mu'} = \chi_\mu + \chi_{\mu'}$
- $\chi_{\mathrm{triv}} = 1$

Exemple. Nous allons calculer le caractère de la représentation χ_ν qui nous sert d'exemple. Comme c'est une fonction centrale, il s'agit de déterminer sa valeur sur chacune des classes de conjugaison de $\mathfrak{S}_3$.

On a déjà $\chi_\nu(1) = 6$. Il reste à déterminer χ_ν sur les éléments (12) et (123)

$$\text{Or } \nu(12) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ et } \nu(123) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Ainsi, on a entièrement déterminé χ_ν :

	1	(12)	(123)
χ_ν	6	2	0

Définition 18 (table des caractères)**.** On appelle table des caractères d'un groupe un tableau dont les colones sont paramétrées par les classes de conjugaison de ce groupe, et les ligne par ses caractères irréductible. Chaque case contient alors la valeur du caractère irréductible correspondant à sa ligne sur la classe de conjugaison correspondant à sa colonne.

8.2.2 Caractères et Composantes Isotypiques

Théorème 6 (orthonormalité des caractères)**.**

- *Si* χ *est un caractère irréductible,* $(\chi|\chi) = 1$
- *Si* χ *et* χ' *sont deux caractères irréductibles non égaux,* $(\chi|\chi') = 0$

Corollaire 1. *L'ensemble des caractères est une famille orthonormée de l'ensemble des fonctions centrales de* G*, en particulier, c'est donc une famille libre. De plus, l'ensemble des fonctions centrales de* G *est un espace vectoriel de dimension* $|\mathrm{Conj}(G)|$*, d'où* $|\widehat{G}| \leq |\mathrm{Conj}(G)|$

Théorème 7. *Soit* μ *une représentation de* G*, alors pour toute représentation irréductible* ρ*, la multiplicité de* ρ *dans* μ *est* $(\chi_\rho|\chi_\mu)$

Corollaire 2. *Un caractère* χ *est irréductible si et seulement si* $(\chi|\chi) = 1$

8.2.3 Caractère de la Représentation Régulière

Dans cette section, on se place sur la représentation régulière, et nous allons étudier son caractère et obtenir des propriétés intéressantes grâce à ce caractère.

Calcul du Caractère

On rappelle que la représentation régulière de G est donnée sur $k[G]$ par la multiplication à gauche. Ainsi, l'action est donnée sur la base de $k[G]$ par $[\mu(g)](g') = gg'$. Ainsi, la matrice de $\mu(g)$ est une matrice de permutation, de plus cette permutation ne contient aucun point fixe, si $g \neq 1$. Cela permet de calculer le caractère de cette représentation.

Proposition 6. $\chi_{\mathrm{Reg}_G}(g) = \begin{cases} |G| & si\ g = 1 \\ 0 & sinon \end{cases}$

Décomposition en Composantes Isotypiques

D'après ce que l'on a démontré sur les caractères, pour décomposer la représentation régulière en composantes isotypiques, il suffit de calculer $(\chi_\rho|\chi_{\mathrm{Reg}_G})$ pour toute représentation irréductible ρ. Or par ce qui précède, $(\chi_\rho|\chi_{\mathrm{Reg}_G}) = \chi_\rho(1) = \dim V_\rho$

Ainsi, on obtient la décomposition en composantes isotypiques de la représentation régulière.

Théorème 8. $\mathrm{Reg}_G = \bigoplus_{\rho \in \widehat{G}} V_\rho^{\oplus \dim V_\rho}$

De plus, on a $(\chi_{\mathrm{Reg}_G}|\chi_{\mathrm{Reg}_G}) = |G|$, ce qui se traduit, en décomposant sur les caractères irrédcutibles, $\sum_{\rho,\rho' \in \widehat{G}} (\dim V_\rho)(\dim V_{\rho'})(\chi_\rho|\chi_{\rho'}) = |G|$.

Cela permet finalement d'aboutir au théorème suivant :

Théorème 9. $|G| = \sum_{\rho \in \widehat{G}} (\dim V_\rho)^2 = \sum_{\rho \in \widehat{G}} (\chi_\rho(1))^2$

Ce théorème possède un intérêt pratique pour établir les tables de caractères.

8.2.4 Nombre de représentations irréductibles

Nous allons maintenant calculer le nombre exact de caractères, ce qui nous permettra de connaître le nombre représentations irréductibles.

Théorème 10. *Les caractères irréductibles forment une base orthonormée de l'espace des fonctions centrales*

Corollaire 3. *Il y autant de représentations irréductibles de G que de classes de conjugaison dans G*

Muni de ce dernier résultat, nous avons tous les outils pratiques pour appliquer la théorie des représentations aux groupes symétriques. Il s'agit d'éléments qui sont déjà présent dans MathComp, et que nous allons beaucoup utiliser. La théorie des caractères, tout particulièrement va nous être d'une grande aide, car elle est formalisée sous une forme facile à manipuler. Nous allons donc maintenant

calculer les représentations des groupes symétriques, en présentant la théorie en parallèle avec sa formalisation en Coq/SSReflect.

Quatrième partie

Representations des Groupes Symétriques

Chapitre 9

Calculs Préliminaires

9.1 Calculs de Tables de Caractères : $\mathfrak{S}_3$, $\mathfrak{S}_4$

Dans un premier temps, nous allons calculer les représentations des groupes symétriques de petits degrés, sans développer de théorie particulière. Cela permettra de nous familiariser avec les techniques de calculs qui nous seront utiles pour la suite. Ces calculs ont été formalisés pour $\mathfrak{S}_2$ et $\mathfrak{S}_3$ dans les fichier ReprS2.v et ReprS3.v. Nous ne présenteront pas ici cette formalisation par soucis de concision. Elle n'a en effet pas d'importance pour la suite mais nous a permis de prendre en main la bibliothèque sur les aspect concernant les représentations, de nous rendre compte quels étaient les démarches les plus pratiques à formaliser, et a donc orienté notre choix d'une démarche pour le cas général.

9.1.1 Table de Caractère de $\mathfrak{S}_3$

Remarquons que pour tout groupe, on connait déjà la représentation triviale, qui est irréductible, et dont le caractère vaut 1 sur toutes les classes de conjugaisons. De plus, pour les groupes symétriques, on dispose d'un autre caractère connu, qui est la signature. En effet, la signature peut être vue comme un morphisme de $\mathfrak{S}_n \longrightarrow \mathbb{C}^* \simeq \mathrm{Aut}(\mathbb{C})$. Cela permet de remplir deux lignes de la table des caractères de $\mathfrak{S}_3$, et on dispose donc de la table suivante.

	1	(12)	(123)
χ_{triv}	1	1	1
χ_{ε}	1	-1	1

Comme le nombre de caractères irréductibles est égale au nombre de classes de conjugaison, on sait qu'il reste une ligne à remplir dans ce tableau. De plus la somme des carrés des nombres de la première colonne doit valoir $|\,\mathfrak{S}_3\,| = 6$, donc la représentation restante est nécessairement de dimension 2. Il ne reste plus qu'à compléter le tableau par orthogonalité des colonnes. On obtient alors la table de caractères de $\mathfrak{S}_3$

	1	(12)	(123)
χ_{triv}	1	1	1
χ_ε	1	-1	1
χ	2	0	-1

Remarquons que pour notre représentation d'exemple ν, on a : $\chi_\nu = 2\chi_{\text{triv}} + 2\chi$, ce qui correspond bien à la décomposition en composantes isotypiques que l'on avait trouvée

9.1.2 Table des Caractères de $\mathfrak{S}_4$

Pour le groupe $\mathfrak{S}_4$, on a toujours les deux caractères χ_{triv} et χ_ε, on va donc compléter la table de caractères suivante :

	1	(12)	(123)	(1234)	(12)(34)
χ_{triv}	1	1	1	1	1
χ_ε	1	-1	1	-1	1

On peut dès à présent remplir la première colonne, en utilisant la propriété sur la somme des carrés. On cherche ainsi trois nombres dont la somme des carrés est 22. L'unique possibilité est donc $(2, 3, 3)$.

Procédons par analogie avec $\mathfrak{S}_3$ et définissons une représentation de $\mathfrak{S}_4$ sur $\mathbb{C}^4$ par permutation des variables. On peut calculer le caractère de cette représentation

	1	(12)	(123)	(1234)	(12)(34)
χ	4	2	1	0	0

Comme dans le cas de $\mathfrak{S}_3$, on peut remarquer que la droite $k \cdot (1, 1, 1, 1)$ est stable sous l'action de $\mathfrak{S}_4$, et que de plus l'action de $\mathfrak{S}_4$ est triviale sur cette droite. Ainsi, cette représentation se décompose en triv $\oplus \rho_1$. De plus le caratère de ρ_1 est donné par $\chi_{\rho_1} = \chi - \chi_{\text{triv}}$. Ainsi, on obtient

	1	(12)	(123)	(1234)	(12)(34)
χ_{ρ_1}	3	1	0	-1	-1

On peut alors vérifier que $(\chi_{\rho_1}|\chi_{\rho_1}) = 1$, ce qui montre que la représentation correspondante est irréductible.

Il reste donc deux représentations irréductibles à déterminer, on peut en trouver une en regardant la représentation de dimension 3 $\rho_2 = \varepsilon\rho_1$. Son caractère est donné par $\chi_{\rho_2} = \chi_\varepsilon\chi_{\rho_1}$, on obtient donc :

	1	(12)	(123)	(1234)	(12)(34)
χ_{ρ_2}	3	-1	0	1	-1

Et à nouveau, on a $(\chi_{\rho_2}|\chi_{\rho_2}) = 1$, donc cette représentation est également irréductible.

Il reste à trouver une représentation irréductible de dimension 2, dont on peut calculer le caractère par orthogonalité des colonnes.

	1	(12)	(123)	(1234)	(12)(34)
χ_{triv}	1	1	1	1	1
χ_{ε}	1	-1	1	-1	1
χ_{ρ_1}	3	1	0	-1	-1
χ_{ρ_2}	3	-1	0	1	-1
χ_{ρ_3}	2	0	-1	0	-2

9.2 Quelques Constructions sur les Représentations

Nous allons maintenant construire les représentations irréductibles pour n'importe quel groupe symétrique en nous intéressant tout particulièrement à la structure spécifique de cette famille de groupes, et en donnant un procédé pour créer des représentations de grandes dimension, à partir de représentations de petites dimensions. Le produit tensoriel des représentations n'est pas formalisé tel quel dans la bibliothèque MathComp, et donc nous en donnons une définition, formalisée dans le fichier towerSn.v

9.2.1 Produit Tensoriel

Une notion que l'on peut introduire pour l'étude des groupes symétriques est le produit tensoriel, qui étant donnés deux groupes G, H munis respectivement d'une représentation (μ, V_μ) et (ρ, V_ρ), permet de construire une représentation de $G \times H$.

Définition 19. On appelle produit tensoriel de μ et ρ, et l'on note $\mu \otimes \rho$, la représentation de $G \times H$ sur $V_\mu \otimes V_\rho$ donnée par : $[\mu \otimes \rho(g,h)](v \otimes w) = [\mu(g)](v) \otimes [\rho(h)](w)$

```
Variables (n1 n2 : nat).
Variables (rG : mx_representation algCF G n1)
          (rH : mx_representation algCF H n2).

Lemma extprod_mx_repr : mx_repr (setX G H) (fun x => tprod (rG x.1) (rH x.2)).
Definition extprod_repr := MxRepresentation extprod_mx_repr.
```

On peut facilement exprimer le caractère de la représentation produit en fonction des caractères des représentations initiales :

Proposition 7. $\forall (g,h) \in G \times H,\ \chi_{\mu\otimes\rho}(g,h) = \chi_\mu(g)\chi_\rho(h)$

```
Section CFExtProdDefs.

Variables (gT aT : finGroupType).
Variables (G : {group gT}) (H : {group aT}).
```

```
Local Open Scope ring_scope.

Lemma cfextprod_subproof (g : 'CF(G)) (h : 'CF(H)) :
  is_class_fun <<setX G H>> [ffun x => (g x.1) * (h x.2)].
Definition cfextprod g h := Cfun 0 (cfextprod_subproof g h).

End CFExtProdDefs.

Notation "f \ox g" := (cfextprod f g) (at level 40, left associativity).

Variables (n1 n2 : nat).
Variables (rG : mx_representation algCF G n1)
          (rH : mx_representation algCF H n2).

Lemma cfRepr_extprod : cfRepr extprod_repr = cfRepr rG \ox cfRepr rH.
```

9.2.2 Induction et Restriction

Nous allons maintenant étudier le lien entre les représentation d'un groupe et celles de ses sous-groupes. Pour cela, nous allons considérer un groupe G et un sous-groupe $H \subset G$. Ces deux notions, ainsi que la réciprocité de Frobenius qui les lient entre elle sont formalisées dans MathComp, et constituent donc un outil à disposition.

Représentation Restreinte

On suppose ici, que d'une représentation (μ, V_μ) de G. On peut alors sans difficulté construire une représentation de H qui provient de la représentation choisie sur G.

Définition 20. On appelle restriction de la représentation μ à H, la représentations de H obtenue sur l'espace vectoriel V_μ en restreignant le morphisme μ aux éléments de H. On note $\mathrm{Res}^G_H \mu$ cette restriction, et formellement, on a : $\mathrm{Res}^G_H \mu := (\mu|_H, V_\mu)$

De la même manière, on peut définir l'opération de restriction sur les fonctions centrales.

Définition 21. Soient φ une fonctions centrale, de G, sa restriction à H, notée $\mathrm{Res}^G_H \varphi$ est définie par $\forall h \in H\, \mathrm{Res}^G_H \varphi(h) = \varphi(h)$. On obtient alors une fonction centrale sur H.

Il est maintenant très simple de vérifier que les deux opérations de restriction sont compatible, modulo le passage aux caractères :

Proposition 8. $\chi_{\mathrm{Res}^G_H \mu}(h) = \mathrm{Res}^G_H \chi_\mu(h)$

En d'autres termes, le caractère de la restriction est la restriction du caractère initial sur les classes de conjugaisons de G qui sont représentées dans H.

Représentation Induite

On dispose également d'une opération duale de la restriction, qui étant donné une représentation (μ, V_μ) de H va nous donner une représentation de G.

Définition 22. L'induction de la représentations V_μ sur G, notée $\mathrm{Ind}_H^G \mu$ est la représentation $\mathrm{Reg}_G \otimes_H V_\mu$, avec le morphisme défini par $\forall x \in k[G],\ v \in V_\mu, [\mathrm{Ind}_H^G \mu(g)](x \otimes v) = (g.x) \otimes v$

Encore une fois, on peut également définir l'induction sur les fonction centrales, de la manière suivante

Définition 23. Soit φ une fonction centrale de H, la fonction centrale induite par φ sur G, notée $\mathrm{Ind}_H^G \varphi$ est définie par $\forall g \in G, \mathrm{Ind}_H^G \varphi(g) = \frac{1}{|H|} \sum_{\substack{s \in G \\ s^{-1}gs \in H}} \varphi(s^{-1}gs)$

A nouveau, ces deux définitions sont compatibles.

Proposition 9. $\chi_{\mathrm{Ind}_H^G \mu} = \mathrm{Ind}_H^G \chi_\mu$

Réciprocité de Frobenius

Il existe un lien très fort entre l'induction et la restriction des représentations, il peut être explicité de plusieurs façons, mais pour la formalisation, notre usage, nous auront besoin de sa formulation sur les caractères. Ce théorème est déjà formalisé dans la bibliothèque MathComp, il sera donc admis ici, et considéré comme un outil à disposition.

Théorème 11 (Réciprocité de Frobenius)**.** *Soient $\varphi \in \mathcal{CF}(H)$ et $\psi \in \mathcal{CF}(G)$, on a alors* $(\mathrm{Ind}_H^G \varphi | \psi)_G = (\varphi | \mathrm{Res}_H^G \psi)_H$

Chapitre 10

Application aux Représentations des Groupes Symétriques et Formalisation

Nous allons maintenant appliquer ces procédés de construction aux groupes symétriques en respectant les structures d'inclusion. En particulier, on remarque que le groupe $\mathfrak{S}_m \times \mathfrak{S}_n$ peut être vu comme un sous-groupe du groupe $\mathfrak{S}_{m+n}$, en considérant des permutations qui agissent séparément sur $\{1, ..., m\}$ et sur $\{m+1, ..., n\}$. Cette approche va nous permettre d'obtenir des représentations de $\mathfrak{S}_{m+n}$ en induisant des produits tensoriels de représentations de $\mathfrak{S}_m$ et de $\mathfrak{S}_n$. Il s'agira ensuite d'étudier lesquelles sont irréductibles, et de montrer qu'on obtient ainsi toutes les irréductibles.

10.1 Une Famille de Représentations

L'objectif est maintenant de construire une famille de représentations que nous allons ensuite étudier, afin de montrer qu'il s'agit des irréductibles. Pour cela, nous allons considérer le groupe $\mathfrak{S}_m \times \mathfrak{S}_n$ comme sous groupe de $\mathfrak{S}_{m+n}$. Il s'agit de considérer des représentations que l'on obtient de manière systématique $\mathfrak{S}_m \times \mathfrak{S}_n$, et de les induire sur $\mathfrak{S}_{m+n}$. La représentation que l'on peut considérer sur $\mathfrak{S}_m \times \mathfrak{S}_n$ est la représentation $\mathrm{triv}_{\mathfrak{S}_m} \otimes \mathrm{triv}_{\mathfrak{S}_n}$. Nous allons donc définir sur $\mathfrak{S}_{m+n}$ la représentation $\alpha_{m,n} := \mathrm{Ind}_{\mathfrak{S}_m \times \mathfrak{S}_n}^{\mathfrak{S}_{m+n}} \mathrm{triv}_{\mathfrak{S}_m} \otimes \mathrm{triv}_{\mathfrak{S}_n}$. Il va s'agir ensuite d'étudier les représentations que l'on vient de définir.

10.1.1 Definitions Préliminaires et Notations en Coq

Une Tour de Groupes

Nous allons dans un premier temps décrire la situation précédente en Coq/SSReflect. Il s'agit d'un résultat présent dans le fichier towerSn.v et qui nous sera indispensable pour formuler les propositions qui suivront.

```
Section TowerMorphism.
Variables m n : nat.
Local Notation ct := cycle_typeSN.
```

```
Local Notation Smn := (setX [set: 'S_m] [set: 'S_n]).

Definition tinjval (s : ('S_m * 'S_n)) :=
  fun (x : 'I_(m + n)) => let y := split x in
  match y with
  | inl a => unsplit (inl (s.1 a))
  | inr a => unsplit (inr (s.2 a))
  end.

Fact tinjval_inj s : injective (tinjval s).
Definition tinj s : 'S_(m + n) := perm (@tinjval_inj s).

Fact tinj_morphM : {morph tinj : x y / x * y >-> x * y}.
Canonical morph_of_tinj := Morphism (D := Smn) (in2W tinj_morphM).

(* The image of 'S_m * 'S_n via tinj with a 'S_(m + n) group structure *)
Definition tinj_im := tinj @* ('dom tinj).

Lemma isom_tinj : isom Smn tinj_im tinj.

Lemma cycle_type_tinj s : ct (tinj s) = union_intpartn (ct s.1) (ct s.2).
End TowerMorphism.

Arguments tinj {m n} s.
Arguments tinj_im {m n}.

Notation "f \o^ g" := (cfIsom (isom_tinj _ _) (cfextprod f g)) (at level 40).
```

Dans ce code, `tinj` désigne le morphisme injectif de plongement de $\mathfrak{S}_m \times \mathfrak{S}_n$ dans $\mathfrak{S}_{m+n}$, `tinj_im` est l'image de ce morphisme, c'est à dire le groupe $\mathfrak{S}_m \times \mathfrak{S}_n$ vu comme sous-groupe de $\mathfrak{S}_{m+n}$, et étant donné des représentations f et g respectivement de $\mathfrak{S}_m$ et $\mathfrak{S}_n$, l'expression `f \o^ g` désigne la représentation $f \otimes g$ de `tinj`$\simeq \mathfrak{S}_m \times \mathfrak{S}_n$.

Les Indicatrices de Classes

Nous allons maintenant définir une notation, qui est présente dans le fichier cycletype.v, et qui permet de nommer aisément les fonctions de classes indicatrices pour les classes des groupes symétriques. Comme celles-ci sont paramétrées par les partitions, grâce au type cyclique, on souhaite qu'il en soit de même pour ces fonctions indicatrices. On va donc utiliser la fonction `perm_of_partCT` qui fournit une permutation d'un type cyclique donné.

```
Section CFunIndicator.
Variable tc : intpartn #|T|.

Definition cfuni_part :=
  cfun_indicator [set: {perm T}] (class_of_partCT tc).

Local Notation "''1_[' p ]" := (cfuni_part p) : ring_scope.

Lemma cfuni_partE s :
  ('1_[s]) = (cycle_type s == tc)%:R.
End CFunIndicator.

Notation "''1_[' p ]" := (cfuni_part p) : ring_scope.
```

En pratique, étant donné une partition `p`, on aura la fonction indicatrice du type cyclique `p` notée `'1_[p]`.

10.1.2 Une Base des Fonctions Centrales

Le travail que l'on va maintenant effectuer est de consiste à remarquer que chaque représentation donne une fonction centrale, via le caractère, et à ensuite utiliser quelques résultats théoriques sur les fonctions centrales. En particulier, nous allons choisir une base privilégiée sur laquelle nous allons étudier les opérations de produit tensoriels et d'induction, pour ensuite décomposer les représentations particulières que nous venons de définir. Nous allons présenter ces résultats accompagnés de leur formalisation qui se trouve dans le fichier towerSn.v. Les notations que l'on va définir dans cette section ne sont pas standards mais se rapprochent de celles que l'on souhaite définir en Coq.

La base que nous allons choisir est en fait une renormalisation de la base des $(\mathbb{1}_C)$.

Définition 24. Etant donné une classe de conjugaison C du groupe G, on note $\mathbb{P}_C = \frac{|G|}{|C|}\mathbb{1}_C$

```
Definition zcoeff (k : nat) (p : intpartn k): algC := #|'S_k|%:R/#|class_of_partCT p|%:R.
Local Notation "'z_ p " := (zcoeff p) (at level 2).
Definition pbasis (k : nat) (p : intpartn k) := 'z_p *: '1_[p].
Local Notation "''P_[' p ]" := (pbasis p).
```

Nous allons maintenant étudier les différentes propriétés de la base $\mathbb{1}_C$, pour ensuite les transférer sur la base $\mathbb{P}_C$

Proposition 10. *Soit C_1, C_2 des classes de conjugaisons de $\mathfrak{S}_m$ et $\mathfrak{S}_n$, alors $\mathbb{1}_{C_1}\mathbb{1}_{C_2} = \mathbb{1}_{C_1 \times C_2}$*

```
Lemma cfextprod_cfuni p q :
  '1_[p] \ox '1_[q] = '1_(classX p q).
```

10.1.3 Décomposition sur cette Base

Nous allons maintenant étudier la décomposition de n'importe quelle fonction centrale sur notre base

Proposition 11. $f = \sum_{C \in \mathrm{Conj}(G)} \frac{|C| f(C)}{|G|} \mathbb{P}_C$

```
Lemma pbasis_gen k (f : 'CF([set: 'S_k])) :
  f = \sum_(p : intpartn k) f (perm_of_partCT p) / 'z_p *: 'P_[p].
Lemma cfdotr_pbasis k (f : 'CF([set: 'S_k])) x : (f x) = '[f, 'P_[ct x]].
```

En appliquant ce résultat au caractère trivial, c'est à dire à la fonction centrale constante égale à 1, on obtient immédiatement

Proposition 12. *Pour tout* $n \in \mathbb{N}$, $\chi_{\mathrm{triv}_{\mathfrak{S}_n}} = \sum_{C \in \mathrm{Conj}(\mathfrak{S}_n)} \frac{|C|}{|G|} \mathbb{P}_C$

10.2 Calcul des Caractères et Produits Scalaires

Pour la suite, si p est une partition d'un entier k, on note $[p]$ la classe de conjugaison de $\mathfrak{S}_k$ constituée des éléments de type cyclique p.

Proposition 13. *Soit* l *une partition de* $m+n$, *alors on a* $\mathrm{Res}^{\mathfrak{S}_{m+n}}_{\mathfrak{S}_m \times \mathfrak{S}_n} \mathbb{1}_{[l]} = \sum_{p \cup q = l} \mathbb{1}_{[p]} \mathbb{1}_{[q]}$

```
Theorem cfuni_Res (l : intpartn (m + n)):
  'Res[tinj_im] ('1_[l]) =
  \sum_(x | l == union_intpartn x.1 x.2) ('1_[x.1] \o^ '1_[x.2]).
```

En utilisant la réciprocité de Frobenius, on montre ensuite le résultat suivant

Proposition 14. $(\mathrm{Ind}^{\mathfrak{S}_{m+n}}_{\mathfrak{S}_m \times \mathfrak{S}_n}(\mathbb{1}_{[p]}\mathbb{1}_{[q]}) | \mathbb{1}_{[l]}) = \frac{|[p]||[q]|}{|\mathfrak{S}_m \times \mathfrak{S}_n|} \delta_{p \cup q = l}$

```
Lemma cfdot_Ind_cfuni_part p q (l : intpartn (m + n)):
  '['Ind[S] ('1_[p] \o^ '1_[q]), '1_[l]] =
  (union_intpartn p q == l)%:R * #|class p|%:R * #|class q|%:R / #|Smn|%:R.
```

Cela permet finalement de montrer la loi d'induction sur les éléments de la base $(\mathbb{P}_C)$

Proposition 15. $\mathrm{Ind}^{\mathfrak{S}_{m+n}}_{\mathfrak{S}_m \times \mathfrak{S}_n} P_{[p]} P_{[q]} = P_{[p \cup q]}$

```
Theorem Ind_pbasis p q :
  'Ind[S] ('P_[p] \o^ 'P_[q]) = 'P_[union_intpartn p q].
```

10.3 L'isomorphisme de Frobenius et la Correspondance de Robinson-Schensted-Knuth

Nous avons maintenant défini une famille de représentations de $\mathfrak{S}_k$, qui est décrite de la façon suivante : pour toute $(k_1, ..., k_l)$ partition de k, on a la représentation $\text{triv}_{\mathfrak{S}_{k_1}} \otimes ... \otimes \text{triv}_{\mathfrak{S}_{k_l}}$. On obtient ainsi autant de représentations de $\mathfrak{S}_k$ que de partition de k, c'est à dire autant que de classes de conjugaisons de $\mathfrak{S}_k$. Il suffit donc de vérifier que toutes ces représentations sont irréductibles pour aboutir au résultat souhaité. Pour cela il s'agit de calculer la norme des caractères. Comme il s'agit d'un calcul délicat, il est plus simple de passer par une étape intermédiaire. En effet, on dispose d'un isomorphisme dit de Frobenius, qui envoie les fonctions centrales de $\mathfrak{S}_k$ sur les fonctions symétriques. Les indicatrices de classes sont alors envoyés sur les powersum et les caractères irréductibles sur les fonctions de Schur. Ce morphisme transforme de plus l'induction de du produit de deux caractères en simple produit des polynômes. Le calcul devient donc beaucoup plus simple sur ces fonctions, et en exprimant tout les termes, notamment la taille des classes de conjugaisons, il ne reste qu'une identité à montrer, qui est donnée par la correspondance de Robinson-Schensted-Knuth. Il s'agit d'une égalité donnée par une bijection entre deux ensembles. Ces résultats ont été partiellement formalisés par Florent Hivert dans le fichier frob.v ainsi que dans le fichier RSK.v du projet Coq-Combi, sur la branche RSK.

Conclusion

Ce stage a été l'occasion de mettre en applications de nombreuses connaissances acquises au cours de l'année, et d'explorer tout particulièrement les liens qui peuvent exister entre l'informatique et les mathématiques. Si les débuts ont demandés beaucoup d'apprentissage, afin de se familiariser avec la combinatoire élémentaire des groupes symétriques et l'extension SSReflect, ils m'ont également permis d'observer la méthode de travail à adopter, et d'acquérir une certaine rigueur dans mon organisation. La seconde phase de ce stage a consisté en l'étude des différents moyens de calculer les représentations des groupes symétriques. En particulier, une grande partie du temps a été consacrée à l'approche de Vershik et Okounkov [VO05, CSST10], qui s'est finalement révélée inutilisable en pratique, avec les outils présents dans MathComp. Après réflexion, nous avons opté pour une formalisation de la méthode développée par Prasad [Pra15] et la fin du stage a été dédiée à formaliser cette méthode. Avec un peu d'expérience, nous avons décidé de transposer cette méthode aux caractères et d'utiliser l'isomorphisme de Frobenius afin de se ramener aux fonctions symétriques pour utiliser la correspondance de Robinson-Schensted-Knuth.

Au delà du travail réalisé, j'ai également découvert le fonctionnement d'une équipe en interne, et eu un aperçu de nombreux sujets de combinatoire grace aux séminaires réguliers qui se tenaient au LIX. Cela m'a permis d'appréhender mieux les différentes problématiques dans lesquelles s'inscrivent mon projet.

De nombreux prolongements sont envisageables pour ce projet. Le plus immédiat est de terminer la formalisation des représentations des groupes symétriques en faisant intervenir la correspondance de Robinson-Schentsted-Knuth. Mais il est également envisageable d'adapter toute la démarche pour formaliser des calculs de représentations d'autres groupes finis ou des représentations monoïdes. Il est aussi envisageable de formaliser d'autres résultats de combinatoires plus poussés qui reposent sur les représentations du groupe symétrique. Enfin, d'une façon beaucoup général, MathComp est une bibliothèque très spécifique qui formalise des résultats très poussés dans un domaine particulier, mais il est toujours intéressant de tenter d'appliquer une telle approche à d'autres théories mathématiques.

Bibliographie

[BGBP08] Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical big operators. *Theorem Proving in High Order Logics*, 2008.

[CSST10] Tullio Ceccherini-Silberstein, Fabio Scarabotti, and Filippo Tolli. *Representation Theory of the Symmetric Groups - The Okounkov-Vershik Approach, Character Formulas and Partition Algebras*. Cambridge studies in advanced mathematics, 2010.

[FH91] William Fulton and Joe Harris. *Representation Theory - A First Course*. Springer, 1991.

[GMT15] Georges Gonthier, Assia Mabhoubi, and Enrico Tassi. A small scale reflection extension for the coq system. Technical report, Inria, 2015.

[Pra15] Amritanshu Prasad. *Representation Theory - A Combinatorial Viewpoint*. Cambridge studies in advanced mathematics, 2015.

[Ser71] Jean-Pierre Serre. *Linear Representations of Finite Groups*. Springer, 1971.

[Ser16] Ilya Sergey. Programs and proofs - mechanizing mathematics with dependent types. Lecture Notes, 2016.

[VO05] Anatoly Vershik and Andrei Okounkov. A new approach to the representation theory of the symmetric groups. ii. 2005.