Formalisation et validation d'une méthode de construction de systèmes de blocs*

Jessy Colonval† et Henri de Boutray‡
Institut FEMTO-ST, Université de Bourgogne Franche-Comté, CNRS, Besançon, France

Résumé

Dans le domaine des mathématiques expérimentales, les programmes utilisés pour obtenir divers résultats suivent rarement les bonnes pratiques du génie logiciel, ce qui rend ces résultats difficiles à reproduire et à évaluer. Cet article présente une formalisation d'une méthode de construction de structures combinatoires (appelées "systèmes de blocs") et une validation de leurs propriétés.

1 Introduction

Dans le domaine des mathématiques expérimentales, les chercheurs développent des programmes informatiques pour découvrir de nouveaux résultats. Cependant, les publications scientifiques de ces résultats ne sont pas toujours accompagnées par ces programmes, ou sinon ces derniers ne sont pas suffisamment accessibles, structurés et documentés pour permettre aux lecteurs de les exécuter pour reproduire ces résultats. Or, la reproductibilité est un enjeu scientifique majeur, qui doit aussi s'appliquer à la partie calculatoire des mathématiques [SBB+12], en particulier pour faciliter la vérification et l'extension des résultats publiés.

Un théorème mathématique est souvent un énoncé général qui doit être démontré, car le calcul ne permet de valider qu'un fragment limité d'une infinité de cas couverts par cet énoncé. Ce théorème peut être vérifié par un tiers, dans un assistant de preuve, si sa publication est accompagnée par une démonstration formelle. Cependant, la construction d'une démonstration formelle est une tâche ardue, car elle nécessite une formalisation de tous les concepts mathématiques sous-jacents au théorème, et la maîtrise d'un assistant de preuve. La démonstration formelle du *odd order theorem* [GAA+13] en est un bon exemple : elle a mobilisé de nombreux chercheurs sur une longue période. Comparativement, le test d'un grand nombre d'instances d'un théorème est souvent beaucoup plus simple à mettre en œuvre et peut apporter un niveau de confiance suffisant dans la correction de ce théorème.

Nous proposons d'appliquer aux mathématiques calculatoires le test intensif, ainsi que d'autres bonnes pratiques de programmation, comme la structuration, la factorisation, la documentation et la distribution libre du code source. Nous visons en particulier les théorèmes portant sur une infinité de structures mathématiques munies d'une taille. Dans ce cas, nous proposons de vérifier le théorème pour toutes les structures, par taille croissante, jusqu'à une taille maximale jugée suffisante pour donner confiance en la correction du théorème.

*Financé par le projet ISITE-BFC I-QUINS (contrat ANR-15-IDEX-03) du programme français "Investissements d'Avenir".

†Master recherche.

‡Doctorant.

Ce travail s'inscrit dans le cadre de recherches des géométries finies dites *quantiques*, car liées à la contextualité quantique. Planat et al. [PGHS15] ont montré comment construire ces géométries à partir de groupes de permutations, mais sans publier de programme pour cette construction. Ces géométries sont des cas particuliers de *systèmes de blocs* (en anglais *block designs*, définis dans la partie 2) et l'article de Planat et al. présente une méthode pour les construire.

Une recherche bibliographique sur l'origine de cette méthode nous a menés à une référence antérieure [KM02], qui présente une méthode plus simple, qui construit des systèmes de blocs à partir de groupes de permutations primitifs. Cet article est complété par un programme, mais ce dernier ne contient pas de code pour valider cette méthode. Nous formalisons cette méthode, puis sa validation par énumération. Plus précisément, nous validons que les systèmes de blocs construits selon cette méthode ont tous les caractéristiques annoncées dans une proposition de cet article. Nous utilisons l'environnement Magma [BCP97], composé d'un langage impératif structuré et d'une vaste bibliothèque de fonctions mathématiques, en particulier de la théorie des groupes et des *designs*. Notre code source est distribué librement sur GitHub [1].

La partie 2 présente notre implémentation de la méthode de construction de systèmes de blocs de Key et Moori [KM02]. La partie 3 traite du test intensif de cette méthode.

2 Construction de systèmes de blocs

Key et Moori [KM02] définissent une méthode de construction de systèmes de blocs par la proposition suivante :

> **Proposition 1.** *Let G be a finite primitive permutation group acting on the set Ω of size n. Let $\alpha \in \Omega$, and let $\Delta \neq \{\alpha\}$ be an orbit of the stabilizer G_α of α. If $\mathcal{B} = \{\Delta^g : g \in G\}$ [...] then $\mathcal{B}$ forms a self-dual 1-$(n, |\Delta|, |\Delta|)$ design with n blocks [...].*
>
> (Extrait de la proposition 1 de la page 3 de [KM02])

La partie 2.1 explique cette proposition, mais le lecteur peu familier de ces notions peut en admettre le contenu. La partie 2.2 présente notre implémentation du contenu calculatoire de cette proposition.

2.1 Définitions

La proposition 1 s'applique à tout entier naturel n et à tout groupe de permutations G sur un ensemble fini Ω de cardinalité n, identifié ici à l'ensemble $\{1, \ldots, n\}$. Par nature, ce groupe G est fini. L'application de la permutation g de G à l'élément α de l'ensemble Ω est notée $g \bullet \alpha$. L'image d'une partie Δ de Ω par la permutation g de G est notée $\Delta^g =_{\mathsf{def}} \{g \bullet x \ : \ x \in \Delta\}$. Les définitions suivantes de la théorie des groupes et de la théorie des systèmes de blocs permettent de comprendre la suite de la proposition, dans laquelle $|\Delta|$ désigne la cardinalité de l'orbite Δ.

Le groupe de permutations G sur Ω est *transitif* si, pour tous les éléments x et y de Ω, il existe une permutation g de G telle que $g \bullet x = y$. Le groupe G est *primitif* s'il est transitif et s'il ne préserve aucune partition non triviale de Ω (les partitions triviales de Ω sont la partition $\{\Omega\}$, dont le seul élément est Ω, et la partition $\{\{x\} \ : \ x \in \Omega\}$, dont tous les éléments sont des singletons). Le *stabilisateur* $G_\alpha =_{\mathsf{def}} \{g \in G \ : \ g \bullet \alpha = \alpha\}$ d'un élément α de Ω (*sous l'action de* G) est l'ensemble des permutations de G qui laissent α invariant sous leur action. L'orbite $O_x =_{\mathsf{def}} \{g \bullet x \ : \ g \in H\}$ d'un élément x de Ω selon un groupe H de permutations sur Ω est l'ensemble des images de x par les permutations de H.

1. `https://quantcert.github.io/Designs`

Un *bloc* est une partie de l'ensemble Ω. Un *système de blocs* $\mathcal{B}$ est un ensemble de blocs. Une *structure d'incidence* est un triplet $\mathcal{D} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ où $\mathcal{P} = \{1, \ldots, n\}$ est un ensemble d'éléments fini, $\mathcal{B} = \{1, \ldots, b\}$ numérote un système de blocs sur $\mathcal{P}$ et $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{B}$ est une *relation d'incidence*, qui définit l'appartenance d'un élément à un bloc. Sa structure *duale* est la structure d'incidence $\mathcal{D}^t =_{\mathsf{def}} (\mathcal{B}, \mathcal{P}, \mathcal{I}^t)$ où $(y, x) \in \mathcal{I}^t$ si et seulement si $(x, y) \in \mathcal{I}$. La relation d'incidence permet d'associer un système de blocs à toute structure d'incidence, et de définir le dual d'un système de blocs. Un système de blocs est *symétrique* s'il possède autant d'éléments que de blocs. Il est *auto-dual* (*self-dual* en anglais) s'il est de plus isomorphe à son dual. Un *système de blocs* t-(v, k, λ) est un système de v blocs de cardinalité k, tel que chaque sous-ensemble de λ blocs a exactement t éléments distincts en commun.

La proposition 1 affirme que, pour tout élément α de Ω et pour toute orbite Δ du stabilisateur de α sous l'action de G différente de l'orbite $\{\alpha\}$, le système de blocs $\mathcal{B} =_{\mathsf{def}} \{\Delta^g : g \in G\}$ contient n blocs et présente la régularité d'un système 1-$(n, |\Delta|, |\Delta|)$ auto-dual. L'exemple 1 illustre ces définitions à partir d'un groupe donné.

Exemple 1. *Soit* $n = 5$ *et* $G = \{$ *Id,* $(1,2,3,4,5)$*,* $(1,5,4,3,2)$*,* $(1,3,5,2,4)$*,* $(1,4,2,5,3)$*,* $(2,5)$ $(3,4)$*,* $(1,2)$ $(3,5)$*,* $(1,5)$ $(2,4)$*,* $(1,3)$ $(4,5)$*,* $(1,4)$ $(2,3)$ $\}$ *un groupe de permutations primitif sur* $\Omega =_{\mathit{def}} \{1,2,3,4,5\}$*. Outre la permutation identité, notée Id, ses permutations sont données sous la forme de leur produit de cycles disjoints. Les points fixes, tels que le cycle* (1) *pour la sixième permutation, ne sont pas écrits. Avec* $\alpha = 1$*, le stabilisateur de G sur* α *est le groupe* $G_1 = \{Id, (2,5)(3,4)\}$*. Les orbites de* G_1 *sont les ensembles* $\{1\}$*,* $\{2,5\}$ *et* $\{3,4\}$*. Pour* $\Delta = \{2,5\}$*, le système de blocs est* $\mathcal{B} = \{\{2,5\},\{1,3\},\{1,4\},\{2,4\},\{3,5\}\}$*. C'est un système de blocs* 1-$(5,2,2)$ *sur* 5 *éléments, composé de* 5 *blocs de cardinalité* $k = 2$*, où chaque élément (*$t = 1$*) est présent dans exactement* $\lambda = 2$ *blocs distincts.*

2.2 Implémentation

Magma propose des fonctions permettant de construire des groupes primitifs [BCFS19, `PrimitiveGroups`], des orbites d'un groupe [BCFS19, `Orbits`], le stabilisateur d'un élément selon l'action d'un groupe [BCFS19, `Stabilizer`] et la création d'un système de blocs à partir d'une structure d'incidence [BCFS19, `Design`], mais ne propose pas d'implémentation de la méthode de Key et Moori pour construire des systèmes de blocs à partir d'un groupe de permutations primitif.

L'article [KM02] est complété par un code Magma (dans son annexe) qui ne formalise pas le contenu calculatoire de la proposition 1 par une fonction, mais l'applique seulement à deux cas particuliers de groupes de permutations primitifs. Ainsi, ce code n'est pas structuré. De plus, il ne contient pas d'instruction pour valider la propriété des systèmes construits, annoncée dans la proposition 1. Enfin, les résultats retournés par ce code sont mélangés avec ce dernier, sans être placés dans un bloc de commentaires. Tout ceci rend le code peu convaincant et ré-utilisable.

C'est pourquoi nous proposons une implémentation de la proposition 1 par deux fonctions, reproduites dans les listings 1 et 2. Afin de faciliter leur utilisation, elles sont commentées selon le format Javadoc, car Magma ne propose pas d'outil permettant de générer une documentation [2]. Les variables Magma sont nommées et les fonctions sont structurées de telle sorte que le code ressemble le plus possible au texte de la proposition 1.

Le listing 1 présente une fonction qui calcule l'ensemble des orbites Δ définies dans la proposition 1. Elle prend en paramètre le groupe de permutations primitif G et retourne un tableau associatif `Deltas` des orbites Δ indexées par les α correspondants. Pour tous les éléments α de

2. Il existe cependant un projet GitHub "magdoc" visant à remplir cette fonctionnalité.

l'ensemble $\Omega = \{1, \ldots, n\}$, la boucle calcule dans la variable `Galpha` le groupe stabilisateur G_α et ses orbites dans la variable `orbits`, puis associe à l'index α la séquence des orbites Δ différentes du singleton $\{\alpha\}$. Les orbites Δ sont converties en ensembles grâce à la fonction Magma `IndexedSetToSet` [BCFS19] pour que les résultats soient sous une forme utilisable pour la création de certains objets Magma utiles pour la suite.

```
/**
 * Compute orbits of stabilizers of a primitive group [KM02, Proposition 1].
 *
 * @param G::GrpPerm A primitive group
 * @return Deltas::Assoc An associative array indexed by alpha
 *       and containing the corresponding delta set
 */
AllDelta := function(G)
    n := Degree(G);
    Omega := {1..n};
    Deltas := AssociativeArray();
    for alpha in Omega do
        Galpha := Stabilizer(G, alpha);
        orbits := Orbits(Galpha);
        Deltas[alpha] := { IndexedSetToSet(Delta) : Delta in orbits | Delta ne { alpha } };
    end for;
    return Deltas;
end function;
```

Listing 1 – Fonction de calcul des orbites Δ à partir d'un groupe de permutations primitif.

Le listing 2 présente une fonction qui construit les systèmes de blocs à partir d'un groupe primitif. Elle prend en paramètre un groupe de permutations primitif G et retourne un tableau associatif des systèmes de blocs indexés par l'orbite Δ associée. Les orbites Δ sont construites à l'aide de la fonction précédente, puis chaque système de blocs est construit par l'application de chaque permutation du groupe G à une orbite Δ. Le résultat est associé à l'index Δ dans le tableau associatif `blocks`.

```
/**
 * Builds all block designs from a primitive group [KM02, Proposition 1]
 *
 * @param G::GrpPerm A primitive group
 * @return blocks::Assoc An associative array indexed by delta indexes
 *       and containing corresponding block designs
 */
BlckDsgnsFromPrmtvGrp := function(G)
    Deltas := AllDelta(G);
    blocks := AssociativeArray();
    for alpha in Keys(Deltas) do
        for Delta in Deltas[alpha] do
            blocks[Delta] := { Delta^g : g in G };
        end for;
    end for;
    return blocks;
end function;
```

Listing 2 – Fonction de calcul de systèmes de blocs à partir d'un groupe de permutations primitif.

3 Validation

Une recherche bibliographique a révélé l'existence d'une correction de la proposition 1, par les mêmes auteurs [KM08]. Cette correction ne remet pas en cause le calcul des systèmes de blocs mais seulement leur nature : ils doivent être symétriques, et non pas auto-duaux, comme résumé dans la proposition 2.

Proposition 2. *Let G be a finite primitive permutation group acting on the set Ω of size n. Let $\alpha \in \Omega$, and let $\Delta \neq \{\alpha\}$ be an orbit of the stabilizer G_α of α. If $\mathcal{B} = \{\Delta^g : g \in G\}$ [...] then $\mathcal{D} = (\Omega, \mathcal{B})$ forms a symmetric 1-$(n, |\Delta|, |\Delta|)$ design. [...]*

(Extrait de la proposition 1, page 1 de [KM08])

Cette partie tente de valider les propositions 1 et 2 à l'aide d'un test exhaustif, borné par le nombre de groupes primitifs utilisés. Nous présentons d'abord trois fonctions booléennes qui implémentent les propriétés des systèmes de blocs des propositions 1 et 2.

Le listing 3 présente une fonction booléenne Magma pour caractériser les systèmes de blocs 1-(v, k, λ). Elle convertit l'ensemble d'ensembles `blocks` en une structure d'incidence, pour déterminer avec la fonction Magma `IsDesign` [BCFS19] si c'est un t-système de blocs. Si c'est le cas, elle construit ce système de blocs pour en extraire ses caractéristiques avec la fonction Magma `Parameters` [BCFS19], puis elle vérifie que ces caractéristiques sont bien celles attendues.

```
/**
 * Characterization of t-(v,k,lambda) block designs
 *
 * @param blocks::Set The design blocks
 * @param t::RngIntElt The number of elements distinct in lambda blocks
 * @param v::RngIntElt The number of blocks
 * @param k::RngIntElt The cardinality of blocks
 * @param lambda::RngIntElt The number of blocks contains t elements distinct
 * @return BoolElt Indicates that blocks design is a t-(v,k,lambda) block design
 */
CorrectDesign := function(blocks, t, v, k, lambda)
    incidence := IncidenceStructure<v | blocks>;
    if not IsDesign(incidence, t) then
        return false;
    end if;
    record := Parameters(Design(incidence, t));
    return record`v eq v and record`k eq k and record`lambda eq lambda;
end function;
```

Listing 3 – Fonction caractéristique d'un système de blocs 1-(v, k, λ)

La proposition 1 indique que le système de blocs construit est un système de blocs 1-$(n, |\Delta|, |\Delta|)$ auto-dual. Le listing 4 présente une fonction booléenne Magma pour vérifier cette condition. La fonction a pour paramètres le nombre d'éléments du système de blocs, le système de blocs lui-même et son orbite Δ. Elle retourne `true` si le système de blocs construit est un système de blocs 1-$(n, |\Delta|, |\Delta|)$ (vérifié avec la fonction précédente) et s'il est auto-dual (vérifié avec la fonction Magma `IsSelfDual` [BCFS19].

```
/**
 * Conditions of block designs in [KM02, proposition 1]
 *
 * @param n::RngIntElt The number of points of the design
 * @param blocks::Set The design blocks
 * @param Delta::Set The delta that generated the design blocks
 * @return BoolElt Indicates whether the block design is a 1-(n,|delta|,|delta|) block
 *      design and a block design self-dual
 */
CorrectConstructionKM02 := function(n, blocks, Delta)
    return CorrectDesign(blocks, 1, n, #Delta, #Delta) and IsSelfDual(Design<1, n | blocks>);
end function;
```

Listing 4 – Propriétés des systèmes de blocs de la proposition 1.

Magma implémente une fonction `IsSymmetric` [BCFS19] qui caractérise un système de blocs symétrique, mais selon une définition différente de celle utilisée par Key et Moori [KM02]. La documentation Magma ne le précise pas mais, après quelques tests, il semble que Magma implémente la

définition de symétrie des BIBD. Un *BIBD* (*Balanced Incomplete Block Design*) est un système de blocs 2-(v, k, λ). Un BIBD est symétrique s'il possède autant d'éléments que de blocs [Col10]. Ainsi, Magma ne considère comme symétriques que les systèmes de blocs dont le paramètre t est égal à 2.

Le listing 5 présente deux fonctions booléennes Magma pour vérifier la condition de la proposition 2 selon ces deux interprétations de la définition de symétrie. Les fonctions ont pour paramètres le nombre d'éléments du système de blocs, le système de blocs lui-même et son orbite Δ. La première (resp. seconde) fonction retourne `true` si le système de blocs construit est un système de blocs 1-$(n, |\Delta|, |\Delta|)$ et si c'est un BIBD symétrique (resp. s'il possède le même le nombre de blocs que d'éléments).

```
/**
 * Characterization of 1-(n,|delta|,|delta|) block designs that are BIBD symmetric
 *
 * @param n::RngIntElt The number of points of the design
 * @param blocks::Set The design blocks
 * @param Delta::Set The delta that generated the design blocks
 * @return BoolElt Indicates whether the block design is a 1-(n,|delta|,|delta|) block
 *       design and a BIBD symmetric
 */
CorrectConstructionKM08_MagmaSym := function(n, blocks, Delta)
    if not CorrectDesign(blocks, 1, n, #Delta, #Delta) then
        return false;
    end if;
    return IsSymmetric(Design<1, n | blocks>);
end function;

/**
 * Characterization of 1-(n,|delta|,|delta|) block designs with the same
 * numbers of blocks and elements
 */
CorrectConstructionKM08_KMSym := function(n, blocks, Delta)
    if not CorrectDesign(blocks, 1, n, #Delta, #Delta) then
        return false;
    end if;
    record := Parameters(Design<1, n | blocks>);
    return record`b eq record`v;
end function;
```

Listing 5 – Propriétés des systèmes de blocs de la proposition 2 selon les définitions de symétrie de Magma et de Key et Moori.

Le listing 6 présente une fonction de validation des propositions 1 et 2. Magma propose une base de données composée de tous les groupes de permutations primitifs de degré [3] inférieur ou égal à 2 500, soit 16 916 groupes, et de 7 643 groupes primitifs de degré plus élevé. Ces groupes contiennent entre 1 et 4 095 permutations. La fonction Magma `PrimitiveGroups` [BCFS19] retourne la séquence de ces groupes primitifs, triés par degré croissant, puis par cardinalité (nombre de permutations) croissante. Le code parcourt cette séquence et calcule tous les systèmes de blocs à partir du groupe courant grâce à la fonction présentée dans le listing 2. Pour chaque système de blocs, le code vérifie qu'il respecte les conditions des propositions 1 et 2, avec les deux interprétations de la symétrie pour la proposition 2. Les résultats sont enregistrés dans un fichier avec le nombre d'éléments n, l'index du groupe parent dans la séquence, l'orbite Δ associée et le temps de construction du système de blocs testé. Les calculs ont été effectués sur le calculateur du Mésocentre de calcul de Franche-Comté, pour les 74 premiers groupes primitifs, en 574 107 secondes (environ 7 jours), jusqu'au degré $n = 14$ inclus. C'est le degré maximal atteignable dans le délai d'utilisation du mésocentre, limité à 8 jours.

3. Le degré d'un groupe de permutations sur un ensemble fini Ω est la cardinalité de cet ensemble Ω.

Ces calculs peuvent être reproduits librement pour les 48 premiers groupes primitifs, jusqu'au degré $n = 10$, sur le calculateur en ligne de Magma [4], dont l'usage est limité à 120 secondes.

```
/**
 * Validation of Proposition 1 in [KM02] and [KM08]
 *
 * @param nbGrp::RngIntElt Number of smallest first primitive groups tested
 */
procedure verifProp(nbGrp)
    allG := PrimitiveGroups(:Warning := false);
    assert nbGrp le #allG;

    printf "degree,numGrp,delta,isKM02,isKM08,isKM08_v2,time\n";
    for numGrp := 1 to nbGrp do
        G := allG[i];
        n := Degree(G);

        beginBlck := Realtime();
        allBlck := BlckDsgnsFromPrmtvGrp(G);
        tBlck := Realtime(beginBlck);
        for Delta in Keys(allBlck) do
            block := allBlck[Delta];
            printf "%o,%o,%o,%o,%o,%o,%o,%o\n",
                n, numGrp, Sprintf("\"%o\"", Delta),
                CorrectConstructionKM02(n, block, Delta),
                CorrectConstructionKM08_MagmaSym(n, block, Delta),
                CorrectConstructionKM08_KMSym(n, block, Delta),
                tBlck;
        end for;
    end for;
end procedure;
```

Listing 6 – Code implémentant un test exhaustif borné des propositions 1 et 2.

À partir des 74 premiers groupes de permutations primitifs, ce programme construit 926 systèmes de blocs, soit tous les systèmes de blocs possibles pour tous les groupes de permutations primitifs de degré inférieur ou égal à 13 et deux groupes de degré 14. Le programme n'a trouvé aucun contre-exemple pour les propositions 1 et 2. En revanche, il a permis d'exhiber 398 contre-exemples pour l'interprétation erronée de la proposition 2, un nombre élevé pour une différence minime entre les deux définitions de la symétrie. Le plus petit contre-exemple est le groupe symétrique S_2 avec $\alpha \in \{1, 2\}$.

Exemple 2. *Soit* $S_2 = \{Id, (1, 2)\}$ *le groupe symétrique sur* $\Omega = \{1, 2\}$ *et* $\alpha = 1$. *Le stabilisateur de* S_2 *de l'élément* 1 *est le groupe* $G_1 = \{Id\}$ *et les orbites de ce groupe sont les ensembles* $\{1\}$ *et* $\{2\}$. *La seule orbite* Δ *possible est l'ensemble* $\{2\}$. *Le système de blocs construit est* $\mathcal{B} = \{\Delta^{Id}, \Delta^{(1,2)}\} = \{\{2\}^{Id}, \{2\}^{(1,2)}\} = \{\{2\}, \{1\}\}$. *Le système de blocs* $\mathcal{B}$ *est composé de* 2 *blocs de taille* 1 *et* 1 *élément distinct se trouve dans exactement* 1 *bloc, donc c'est un système de blocs* 1-$(2, 1, 1)$. *Ce n'est pas un système de blocs* 2-(v, k, λ), *ce qui le rend non symétrique selon Magma. Mais il est symétrique selon la définition de Key et Moori, car il possède autant d'éléments que de blocs.*

4 Conclusion

Nous avons donné un exemple d'application de la programmation et du test intensif à la vérification de propriétés mathématiques. Nous avons appliqué le test exhaustif borné à deux propositions de construction de systèmes de blocs, selon deux interprétations différentes, ce qui nous a permis de

4. `http://magma.maths.usyd.edu.au/calc/`

valider ces propositions et de lever une ambiguïté sur leurs interprétations possibles. Outre la documentation du code, nous avons aussi identifié une bonne pratique : la proximité entre le programme et le théorème qu'il formalise rassure le lecteur sur la cohérence entre les deux.

Une perspective globale serait d'établir et de diffuser d'autres principes généraux pour la reproductibilité et la vérification des publications mathématiques de nature calculatoire, mais ceci n'entre pas dans le cadre de nos sujets de master et de thèse. La suite du travail de master sera d'appliquer ces méthodes à l'article de Planat et al. [PGHS15]. La suite du travail de thèse sera d'intégrer cette contribution dans nos recherches sur les spécifications des langages quantiques.

Remerciements

Nous remercions Alain Giorgetti, Pierre-Alain Masson et Frédéric Holweck, notre encadrant de stage de master recherche et nos directeur, co-directeur et co-encadrant de thèse, qui nous ont aidé et conseillé tout au long de ce travail. Nous remercions aussi Michel Planat qui nous a aidé à comprendre certaines notions complexes.

Références

[BCFS19] W. Bosma, J. Cannon, C. Fieker, and A. Steel. Handbook of Magma functions, edition 2.24-6, 2019.

[BCP97] W. Bosma, J. Cannon, and C. Playoust. The Magma Algebra System I : The User Language. *Journal of Symbolic Computation*, 24(3) :235–265, 1997.

[Col10] C.J. Colbourn. *CRC Handbook of Combinatorial Designs*. CRC Press, 2010.

[GAA+13] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A Machine-Checked Proof of the Odd Order Theorem. In *Interactive Theorem Proving*, volume 7998, pages 163–179. 2013.

[KM02] J.D. Key and J. Moori. Codes, Designs and Graphs from the Janko Groups J_1 and J_2. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 40 :143–159, 2002.

[KM08] J.D. Key and J. Moori. Correction to : Codes, Designs and Graphs from the Janko Groups J_1 and J_2, J.D. Key and J. Moori, JCMCC 40 (2002), 143-159. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 64 :153, 2008.

[PGHS15] M. Planat, A. Giorgetti, F. Holweck, and M. Saniga. Quantum contextual finite geometries from dessins d'enfants. *International Journal of Geometric Methods in Modern Physics*, 12(07) :1550067, 2015.

[SBB+12] V. Stodden, D.H. Bailey, J. Borwein, R.J. LeVeque, and W. Rider. Setting the Default to Reproducible. Technical report, 2012. `http://icerm.brown.edu/topical_workshops/tw12-5-rcem`.