Computational studies of entanglement and quantum contextuality properties towards their formal verification

Henri de Boutray

THÈSE DE DOCTORAT

DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ

École doctorale n° 37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

par

HENRI DE BOUTRAY

Computational studies of entanglement and quantum contextuality properties towards their formal verification

Études calculatoires de l'intrication et de la contextualité quantiques dans la perspective de leur vérification formelle

Thèse présentée et soutenue à Besançon, le 16/12/2021

Composition du Jury :

JEANDEL EMMANUEL	Professeur à l'Université de Lorraine	Président
LÉVAY PÉTER	Maître de conférence à l'Université de Technologie et d'Économie de Budapest	Rapporteur
LUQUE JEAN-GABRIEL	Professeur à l'Université de Rouen	Rapporteur
PORTIER NATACHA	Maître de conférence à l'École Normale Supérieure de Lyon	Examinatrice
GUYEUX CHRISTOPHE	Professeur à l'Université de Bourgogne Franche-Comté	Examinateur
GIORGETTI ALAIN	Maître de conférence HDR à l'Université de Bourgogne Franche-Comté	Directeur de thèse
HOLWECK FRÉDÉRIC	Maître de conférence HDR à l'Université de Bourgogne Franche-Comté	Co-directeur, invité
MASSON PIERRE-ALAIN	Maître de conférence à l'Université de Bourgogne Franche-Comté	Co-encadrant, invité

N° X X X X X X X X X X X X

Acknowledgements

These last three years have been a dramatic change for me, my way of life is very much different from what it was. But I still owe to my parents most of what I am today. I thank them wholeheartedly for all they have done for me, from my birth to putting up with me during the writing of this manuscript. I also thank my brother, with whom I share so many interests, in particular in scientific fields, to the despair of our mother feeling sometimes lonely during our in-depth dinners conversations.

One of the reasons (maybe even the main one) why these three last years saw such a great shift in my way of life is that I met an incredible woman. She is probably the one who helped me the most during hard times, supporting me when my moral was low and getting excited when I showed her my perfectly understandable articles. I'll never be able to thank my wife enough for her patience with me, and the joy she brings me.

I owe special thanks for Alain Giorgetti, my thesis director. It is a privilege to have worked with him. I believe that this role is crucial in the success of a Ph.D., and I do not speak only about an academic success: a Ph.D. is the first step in a career focused around scientific research, and it should motivate the candidate to pursue such a career. And this was very much the case for me. Alain was always available for me, with a rigor that made me want to better myself each time. And I must say, even though my work was far from perfect, I feel that our mutual understanding helped me see my work through his eyes, and perfect it with practice.

Many thanks to Frédéric Holweck too, I am very impressed in his apparent ability of creating time for always new projects. From my point of view where I am always struggling to finish what I started, I am in awe of all what he is able to achieve. In addition to being so proficient, Frédéric was a great help in understanding the complex mathematical basis of quantum computing, always very instructive and on point.

I was lucky enough to have a team of three supervisors during these three years. Pierre-Alain Masson was the third member of this team, and was always fair and wise in his advice. I praise him for having the patience of staying with us and taking the time of understanding all the new notions involved in this thesis, in spite of the subject shifting away from his field of research during those three years.

I want to respectfully thank Péter Lévay and Jean-Gabriel Luque for reviewing this manuscript, they both had a determinant role in some fields of my researches, and it is an honor for me to have them reviewing my work.

I also want to thank Natacha Portier, Christophe Guyeux and Emmanuel Jeandel for participating in the jury on my thesis. I am especially impatient to meet again Emmanuel Jeandel with whom I entered in the world of academic research back in 2018.

During my thesis, I had the opportunity of working with several other students and researchers. I started with Jessy Colonval, who was the main author on my first paper during his internship with us, I am glad to see him carry on with a Ph.D., and I wish him a great success. I then worked with Hamza Jaffali, another Ph.D. student of Frédéric Holweck, who was, in addition of a great coworker, a great chess opponent: I hope to play with him again soon! Another Ph.D. student of Frédéric who I worked with was Grâce Amouzou, we worked together across continents, during the Covid epidemic, nothing can stop us now. Finally, I thank very sincerely Metod Saniga, who was the main author of my most recent paper, and who taught me a ton on quantum geometries. He is someone who I have a very strong respect for, for his incredible efficiency and speed while working.

Several other researchers guided my work. Benoit Valiron and Christophe Chareton both accomplish important steps toward quantum programs and quantum program formal verification. I had the chance of exchanging with them and hope I will have this chance again. Simon Perdrix also had a huge influence on me, supervising my first experience in quantum computing research, and encouraging me with his enthusiasm. Michel Planat also helped me push through on quantum geometry subjects, his knowledge and sympathy are important reason this specific subject is of interest for me.

Finally, I want to thank all my friends who kept me motivated: Alexandre Eid, Titouan Carette and Maria Borichev were my long distance correspondents on keeping up with a thesis; Jean-Philippe Gros (aka de Molette de Morangier) for our many thrilling philosophical debates, and the good times we had; Florent Marie, Arthur Hugeat and Antoine Chevrot for the many gaming nights; and every other friend that I came across these past years.

Contents

French summary: Études calculatoires de l'intrication et de la contextualité quantiques dans la perspective de leur vérification formelle

Motivations

De nos jours, l'informatique quantique est très présente dans les communications publiques. Depuis les initiatives nationales (France, UK, USA, et plus... [Fel21, Pre20, Smi18]), jusqu'à l'intérêt des chercheurs individuels. Cet intérêt provient des énormes promesses de l'informatique quantique [GKS+21], mais également des défis techniques qui donnent lieu à un riche champ de travail. Mais comme pour l'informatique classique auparavant, l'informatique quantique est confrontée à une problématique très humaine : comment faire confiance au programme ?

Cette question est prédominante dans certains domaines vitaux, où toute erreur du programme peut coûter des vies humaines, comme l'aérospatial, les réacteurs nucléaires ou le réseau ferroviaire parisien [CM14, MLL19, INT99, LSP+07]. Mais elle tend à s'étendre à tous les domaines : en tant qu'utilisateur, nous voudrions nous assurer que le site de notre banque est digne de confiance, ce qui implique par exemple le besoin d'un JavaScript vérifié [BRN+20]. Puisque les programmes quantiques semblent avoir des champs d'application très larges, nous voudrions nous assurer dès le départ qu'ils résolvent le problème pour lequel ils ont été conçus. Cela est d'autant plus vrai que, pour beaucoup, l'informatique quantique a un aspect très peu intuitif, ce qui implique en soi un besoin plus important de *correction*, la propriété selon laquelle le programme effectue bien la tâche pour laquelle il a été conçu.

Problématique

Dans ce contexte, l'objet de cette thèse est de préparer le transfert de certaines méthodes de l'informatique classique à l'informatique quantique. Nous décomposerons le titre de cette thèse (Études calculatoires de la contextualité et de la non-localité quantiques dans la perspective de leur vérification formelle) en ses composantes afin de pouvoir les décrire une par une. En commençant par le dernier élément : la vérification formelle d'un programme quantique est pour l'instant un idéal lointain. En effet, en raison de la jeunesse des processeurs quantiques fonctionnels, nous – en tant qu'utilisateurs – manquons de pratique sur les langages quantiques. Cette pratique faciliterait grandement la vérification formelle des programmes classiques. En outre, les processus impliqués dans la conception de processeurs quantiques ne passent pas encore à l'échelle en nombre de qubits, ce qui entraîne la création d'outils de vérification de programmes quantiques sans ordinateurs capables d'exécuter lesdits programmes. C'est la raison pour laquelle nous avons concentré nos efforts de vérification sur des propriétés – la contextualité et la non-localité quantiques – plutôt que sur des programmes quantiques, ces propriétés étant une marche pouvant nous mener à la vérification de programmes. De plus, ce domaine étant très actif, nous avions l'opportunité d'utiliser des outils existants – pour la vérification classique et quantique – pour étudier ces propriétés. L'idée étant que nous préparons le terrain pour la vérification formelle des programmes quantiques en étudiant des propriétés qui pourront ensuite être utilisées pour spécifier des programmes quantiques. C'est pour cette raison que le choix des propriétés est particulièrement important, et que nous avons choisi l'intrication et la contextualité. En effet, ces propriétés, en plus d'être mesurables, ont une part importante dans la recherche sur l'explication de la suprématie quantique, ce qui laisse donc penser qu'elles sont essentielles dans l'élaboration de programmes quantiques.

Il convient de noter que, comme indiqué précédemment, les informaticiens quantiques manquent de pratique en programmation pour les processeurs quantiques, mais cette pratique n'est pas complètement inexistante non plus. En effet, ces dernières années, certains processeurs quantiques ont fait la une de l'actualité [Qua, Por21, Mat21] et, malgré les limitations de ces processeurs, IBM a donné le contrôle de certains de ses processeurs quantiques au public, par le biais de *IBM quantum experience* [IBM]. C'est un outil précieux pour tester des problèmes suffisamment petits, et il a été utilisé pour cette thèse, malgré ses importantes limitations.

Plan

La partie présente de ce manuscrit est un résumé de l'ensemble du manuscrit. Chaque section numérotée de ce résumé représente une partie distincte du manuscrit.

Les notions nécessaires à la compréhension globale de la thèse sont exposées dans la section 1. Cette section est suivie de mes contributions dans les sections 2 et 3. Certaines notions ne sont introduites que lorsque cela est nécessaire, lorsqu'elles ne sont pas pertinentes pour l'ensemble du manuscrit, afin d'éviter de donner au lecteur trop d'informations à la fois.

La section 1 introduit les méthodes formelles classiques puis des éléments d'informatique quantique, telle qu'abordées par un informaticien. Ce dernier point commence par des bases de l'informatique quantique, suivi par les langages utilisés en informatique quantique, quelques notions de vérification de programmes quantiques, et enfin les deux propriétés au cœur de ce manuscrit : l'intrication et la contextualité.

La section 2 présente un simulateur d'exécution de circuits quantiques que j'ai développé pour étudier l'intrication dans l'algorithme de Grover (un algorithme de recherche) et la transformée de Fourier quantique (QFT, une version quantique de la transformée de Fourier discrète). Dans cette section, le lecteur trouvera également la transposition de cette étude à Qiskit, la bibliothèque d'IBM pour la simulation de circuits quantiques fournissant aussi un moyen d'envoyer les circuits localement créés et de les exécuter sur leurs processeurs quantiques, une forme de *cloud computing* quantique.

La section 3 rassemble des travaux sur les géométries finies intéressantes pour leur lien avec la contextualité quantique. Le premier travail consistait à générer des géométries finies - également appelées systèmes de blocs, et vérifier la correction des algorithmes de génération sous-jacents. Le second travail concernait la génération de géométries finies afin d'étudier l'espace symplectique $W(2n-1, 2)$, un espace géométrique modélisant les relations de commutation du groupe de Pauli.

1/ Contexte

Afin de définir la vérification de programmes quantiques, nous devons d'abord commencer par définir certains termes que nous utiliserons, provenant de la vérification de programmes classiques. De manière générale, la vérification de programme peut être classée en deux catégories : la vérification statique et la vérification dynamique. La vérification statique consiste à raisonner sur le code du programme sans l'exécuter, alors que la vérification dynamique se place à l'opposé, en vérifiant les programmes en les exécutant.

Un exemple de vérification statique est la vérification déductive. Elle se base sur les préconditions (conditions d'entrée du programme) et les postconditions (conditions de sortie du programme), et consiste à montrer que les postconditions peuvent être déduites des préconditions.

Deux exemples de vérification dynamique sont le test et l'évaluation d'assertions à l'exécution. Le test consiste à exécuter le programme avec certaines entrées prédéterminées et valider que la sortie du programme correspond bien à une valeur attendue. Cependant, les tests peuvent prendre de nombreuses formes en fonction de ce qui est testé (test unitaire, test d'intégration, test de non régression...). L'*évaluation d'assertion à l'exécution* (Runtime Assertion Checking en anglais, ou plus communément RAC) consiste à ajouter des assertions dans le code même du programme, qui seront exécutées en même temps que le reste du programme et qui contiennent des formules logiques dont la validité doit être vérifiée lors de l'exécution de l'assertion. Ces assertions permettent de s'assurer que le code se comporte correctement, même en production. Cette pratique peut être mise en parallèle avec les mécanismes d'exceptions inclus dans certains langages.

Maintenant que les notions de base de vérification de programme sont posées, penchons nous sur celles d'informatique quantique. L'informatique quantique consiste en l'utilisation de propriétés de la physique quantique afin de construire un processeur pouvant faire des opérations inaccessibles aux processeurs classiques. Pour comprendre comment marche un tel processeur, examinons les briques élémentaires portant l'information dans un ordinateur quantique.

Dans un ordinateur classique, l'information est divisée en bits, pouvant avoir une valeur généralement notée 0 ou 1. Pour un ordinateur quantique, cette information est divisée le plus couramment en bits quantique, aussi appelés qubits, et dont les états de base sont $|0\rangle$ et $|1\rangle$. Mais la différence la plus importante et qu'un qubit peut être une superposition de ces deux états de base, un tel état peut être noté $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ avec $\alpha, \beta \in \mathbb{C}$ et $|\alpha|^2 + |\beta|^2 = 1$. Continuons le parallèle entre l'informatique quantique et l'informatique classique : pour l'informatique classique, plusieurs bits regroupés forment un registre, qui pourrait par exemple prendre la valeur $r = 001101$; à cause de la superposition, une telle concaténation des valeurs n'est pas aussi simple en informatique quantique. Pour résoudre ce problème, nous utilisons l'opérateur $\otimes$ appelé produit tensoriel ou produit de Kronecker. Les états peuvent être notés sous forme vectorielle, et dans ce cas, on aura $|0\rangle = \binom{1}{0}$ et $|1\rangle = \binom{0}{1}$. En notation vectorielle, le produit tensoriel a l'effet suivant :

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \dots & a_{1,n}B \\ \vdots & & \vdots \\ a_{m,1}B & \dots & a_{m,n}B \end{pmatrix}$$

A étant une matrice à m lignes et n colonnes et $a_{i,j}$ étant les coefficients de A.

Ces états sont modifiés par des portes, comme en informatique classique (porte $ET, OU, \ldots$), et ces portes sont modélisées par des matrices unitaires. Un ensemble de portes connu est l'ensemble des portes de Pauli, composé des matrices $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & i \\ -i & 0 \end{smallmatrix}\right)$ et $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$.

Les états peuvent aussi être mesurés, mais cette fois, la mesure est très différente de son équivalent classique. En effet, un état classique peut être observé à tout instant sans perturber l'état global du système. La mesure d'un état quantique est modélisée par une matrice auto-adjointe appelée observable. Un état s'écrivant $|\varphi\rangle = \alpha |a\rangle + \sum_i \beta_i |b_i\rangle$ mesuré par un observable ayant $\{|a\rangle, |b_1\rangle, \ldots, |b_k\rangle\}$ comme base de vecteurs propres aura une probabilité $|\alpha|^2$ d'être projeté sur $|a\rangle$. Dans ce cas, la mesure retournera la valeur propre de l'observable associée à son vecteur propre $|a\rangle$. Cet effet de la mesure est valable pour tous les vecteurs propres de l'observable, il en résulte donc que la mesure est dite *destructive* en quantique: l'état global du système est affecté par la mesure d'un qubit.

Les portes et les mesures peuvent être représentées sous formes de circuits comme montré dans la figure 1. Mais ces circuits possédant d'importantes limitations comme l'absence de boucles sont eux-mêmes abstraits par des langages comme l'assembleur quantique (QASM [SAC⁺06]) permettant de créer des sous-routines. D'autres langages

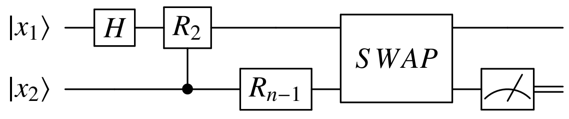


Figure 1: Exemple de circuit quantique

plus riches encore ont été créés afin de générer des circuits plus efficacement comme Qiskit, une librairie python développée par IBM afin de générer des circuits, simuler leur exécution, et même soumettre en ligne les circuits à leurs processeurs quantiques. Enfin, certains chercheurs supposent que la circuiterie n'est tout bonnement pas le bon langage pour les algorithmes quantiques, et des langages sur un concept différent sont proposés, comme le ZX-calcul, un autre langage graphique abstrayant les transformations sous forme de nœud de graphe, et fournissant un riche ensemble de règles de réécriture de graphes.

En s'appuyant sur les principes de vérification précédemment présentés et les concepts de l'informatique quantique, des projets de vérification quantique ont déjà été proposés. Une adaptation de la logique de Hoare à l'informatique quantique a été théorisée [Yin18] même si elle reste peu réalisable en pratique. Des projets concrets ont aussi été publiés comme un outil d'optimisation de circuit vérifiée avec Coq[1] [HRH⁺21] et un outil de preuve

[1] https://coq.inria.fr/

de programmes quantiques avec Why3[2] [CBB+21].

Nous avons choisi pour cette thèse de nous concentrer sur deux propriétés quantiques sur lesquelles la vérification quantique pourrait plus tard s'appuyer. La première de ces propriétés est l'intrication. Un état mettant en évidence l'intrication est l'état de Bell $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Quand on mesure le premier qubit de cet état avec par exemple l'observable $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, le second qubit est projeté en même temps que le premier. Ce résultat contre-intuitif est une des démonstrations les plus élémentaires de l'intrication, que l'on définit formellement en disant que l'intrication est la négation de la séparabilité. Un état est dit séparable s'il peut s'écrire comme le produit tensoriel d'états à un qubit. L'intrication ainsi définie est un concept binaire, mais des raffinements du concept existent, où l'on parle de degré d'intrication. Par exemple, les polynômes de Mermin sont des opérateurs permettant de mettre en évidence un tel degré d'intrication.

La seconde propriété sur laquelle nous nous sommes penchés est la contextualité. Une expérience est dite contextuelle si ses résultats ne peuvent pas être expliqués sans que chaque élément de l'expérience ait une "connaissance" du contexte actuel de l'expérience. Telle que nous l'avons abordée, nous avons surtout étudié la contextualité de géométries finies rattachées à des expériences physiques. Une telle géométrie est un ensemble de points et de lignes contenant des points, et nous la paramétrisons en attribuant à chaque point un observable. Dans ce cas, la géométrie est contextuelle si les résultats des mesures ne peuvent pas être expliqués classiquement sans que les points aient accès aux mesures précédemment effectuées.

2/ Étude de l'évolution de l'intrication au cours de l'algorithme de Grover et de la QFT

Pour étudier l'intrication, nous avons commencé par construire un simulateur, nous permettant ainsi de maîtriser son comportement bien plus finement que si nous avions utilisé un simulateur déjà existant. Des exemples de finesses permises par notre simulateur sont : du calcul exact, un affichage de l'exécution exportable en LaTeX ou encore un format d'entrée personnalisé. Je me suis entraîné sur ce simulateur en validant l'algorithme de Deutsch puis l'algorithme de Deutsch-Jozsa. L'algorithme de Deutsch consiste à prendre une fonction booléenne $f : \mathbb{B} \to \mathbb{B}$ en entrée, et à déterminer en un seul appel de la fonction si elle est constante ou non. L'algorithme de Deutsch-Jozsa quant à lui prend en entrée une fonction booléenne à n entrées $f : \mathbb{B}^n \to \mathbb{B}$ qui doit être équilibrée ($|f^{-1}(\{0\})| = |f^{-1}(\{1\})|$) ou constante ($|f^{-1}(\{0\})| = 0$ ou $|f^{-1}(\{1\})| = 0$), et détermine si elle est équilibrée ou constante en un seul appel de f.

[2]http://why3.lri.fr/

J'ai ensuite utilisé ce simulateur pour valider l'utilisation des polynômes de Mermin [dJH+21] en tant qu'opérateurs de mesure de la non-localité.

Definition 1: Polynômes de Mermin, [ACG+16]

Soit $a = \left(a_j\right)_{j\geq 1}$ et $a' = \left(a'_j\right)_{j\geq 1}$ deux familles d'observables à un qubit avec pour valeurs propres $\{-1, +1\}$. Le polynôme de Mermin $M_n(a, a')$ est défini récursivement par

$$\begin{cases} M_1(a, a') = a_1 \\ \forall n \geq 2,\ M_n = \frac{1}{2}M_{n-1}(a, a') \otimes (a_n + a'_n) + \frac{1}{2}M_{n-1}(a', a) \otimes (a_n - a'_n) \end{cases}$$

Nous avons tout d'abord appliqué ces polynômes à l'algorithme de Grover. En effet, l'algorithme de Grover présente des particularités propres à rendre l'étude de l'intrication des états qu'il traverse particulièrement intéressante. L'algorithme de Grover consiste à trouver un élément dans une liste de N éléments, et arrive à faire ceci en $O(\sqrt{N})$ opérations au lieu de $O(N)$ pour le cas classique. Nous démontrons que l'algorithme part d'un état séparable, et arrive à un état séparable à la fin de son exécution, mais qu'il s'approche au milieu de son exécution d'un état non séparable. La conjecture était donc que l'intrication augmenterait jusqu'à un point proche du milieu de l'exécution, point à partir duquel l'intrication diminuerait. Et c'est bien ce que nous avons observé pour des exécutions entre 4 et 12 qubits. Nous avons ensuite comparé l'évolution de la mesure de l'intrication par les polynômes de Mermin à des résultats obtenus en utilisant l'hyperdéterminant [JH19] pour la QFT (Quantum Fourier Transform). La QFT est une version quantique de la transformée de Fourier discrète. Pour un nombre encodé sur n bits, elle opère cette transformée en $O(n^2)$ opérations contre $O(n2^n)$ pour le cas classique. Cette comparaison a mis en évidence que certaines portes ne changent jamais le niveau d'intrication, ce qui était un résultat attendu, car ces portes sont locales et inversibles. En revanche, un résultat plus surprenant est que, dans certains cas, l'hyperdéterminant et les polynômes de Mermin ne concordaient pas sur l'évolution de l'intrication.

Forts de cette connaissance, nous avons ensuite implémenté ces mêmes calculs sur Qiskit, la bibliothèque de circuiterie d'IBM. Nous avons obtenu des résultats très similaires sur les simulations. En revanche, il semble que le bruit sur leur processeur quantique est encore trop important pour obtenir des résultats aussi fins que ce que nous souhaitions.

Le bilan de cette étude est que les polynômes de Mermin sont bien un moyen d'évaluer l'intrication, et qui plus est, un moyen implémentable physiquement, contrairement à d'autre mesures de l'intrication comme la mesure par l'hyperdéterminant.

3/ Études de la contextualité à l'aide de géométries quantiques

Motivés par des travaux antérieurs réalisés par deux de mes encadrants de thèse, j'ai ensuite abordé l'étude la contextualité du point de vue des géométries finies.

Dans cette section je me focalise sur les géométries dites quantiques, dans le but d'étudier la contextualité. Je commence par étudier une méthode de génération de géométries établie par Key et Moori [KM02], en expliquant le lexique spécifique à ce domaine. Cette méthode avait été implémentée en Magma dès l'article initial, mais le code correspondant était très peu fonctionnel. De plus, un erratum mentionnait que la méthode initialement présentée serait fausse, nous avons donc ré-implémenté le code en question, pour réaliser que la différence observée correspondait en fait à une erreur dans l'interprétation de certaines notions. Pour vérifier tout cela, nous avons appliqué des méthodes de vérification automatisée et de génie logiciel qu'il serait probablement bon de transmettre aux mathématiques expérimentales pour éviter les erreurs telles que celle que nous avons observée.

Une fois cette première expérience sur Magma acquise, nous avons implémenté des méthodes de génération de géométries fondées sur les espaces symplectiques. Ces espaces permettent d'encoder les opérateurs de Pauli efficacement, ainsi que leur relations de commutation. Pour cela, nous nous plaçons dans un espace projectif dont le corps des coefficients a deux éléments. Nous notons les points $p = [0, 1, 1, 1, 0, 0, 1, 0]$, par exemple. L'exemple précédent peut être vu comme la représentation d'un opérateur de Pauli, en prenant l'équivalence suivante : $0, 0 \to I; 0, 1 \to X; 1, 0 \to Z; 1, 1 \to Y$. Dans ce cas, on obtient $p \to X \otimes Y \otimes I \otimes Z$. La relation de commutation entre deux opérateurs est encodée dans la forme symplectique $f(p, q) = \sum_{i \in [0..(n-1)/2]} p_{2i}q_{2i+1} + p_{2i+1}q_{2i}$: si la forme symplectique de deux points est nulle, alors ils commutent. En utilisant cela, nous pouvons construire des géométries complexes, comme l'ensemble des lignes de l'espace symplectique, ou des sous géométries définies par des formes quadratiques. Nous évaluons la contextualité des géométries construites en montrant qu'un système linéaire correspondant n'a aucune solution. Les résultats de ces calculs semblent indiquer des résultats vrais pour tout nombre n de qubits, il serait donc intéressant d'essayer de prouver ces conjectures.

Enfin, nous utilisons des fonctions de génération de géométries de l'espace symplectique pour mettre en évidence la structure des espaces symplectiques. Pour cela, nous développons une fonction calculant la signature d'une géométrie. Signature que nous avons nous-mêmes définie comme étant le $(n + 1)$-uplet – où n est le nombre de qubits de l'expérience – contenant le nombre de lignes négatives de la géométrie, et le nombre d'opérateurs contenant i fois l'identité, pour tout i entre 0 et $n-1$. L'espace symplectique à

n qubits contient des géométries qui sont toutes identiques si on ne prend pas en compte leur paramétrisation, mais qui sont différenciées par cette signature. Par exemple, les formes quadratiques générant des géométries se classent en deux types, elliptiques ou hyperboliques. Pour un type donné, toutes les géométries sont isomorphes si on ignore la valeur des points. En revanche, leurs signatures étant différentes, elles ne sont pas isomorphes quand on garde la valeur des points. En partant de ce constat, nous avons remarqué que certaines géométries de l'espace symplectique à n qubits peuvent générer des géométries de l'espace symplectique à $n+1$ qubits. Plusieurs algorithmes sont proposés prenant en compte différentes géométries.

Conclusion

La vérification des programmes est un élément clé pour leur bon fonctionnement. À ce jour, la vérification des programmes quantiques n'en est qu'à ses débuts. Dans cette thèse, j'ai exploré des manières de spécifier et d'évaluer deux propriétés quantiques – l'intrication et la contextualité – préparant leur formalisation dans un outil de vérification de programmes quantiques.

Dans un premier temps, un simulateur de circuit quantique a été développé dans SageMath, permettant d'évaluer les propriétés des états quantiques à n'importe quelle étape d'exécution intermédiaire, selon la méthode de vérification connue sous le nom de "*runtime assertion checking*". Ce prototype offre également la possibilité de choisir entre le calcul en virgule flottante et le calcul exact, pour des résultats plus rapides ou plus précis. Avec ce simulateur j'ai expliqué deux exemples d'algorithmes – l'algorithme de Deutsch et l'algorithme de Deutsch-Jozsa. L'algorithme de Deutsch a été validé à l'aide de tests exhaustifs – une méthode où toutes les entrées possibles sont testées. L'algorithme de Deutsch-Jozsa a été validé jusqu'à $n = 5$ qubits en utilisant le test exhaustif borné. Ces deux méthodes sont des moyens très élémentaires de valider un logiciel, disponibles uniquement pour certains cas simplistes. Ce même simulateur a été utilisé afin d'étudier la variation de l'intrication dans l'algorithme de Grover et la QFT, en utilisant les polynômes de Mermin [dJH+20, dJH+21]. Dans ce travail, nous avons conclu que les polynômes de Mermin sont un outil approprié pour mesurer dynamiquement l'intrication à chaque étape d'un algorithme quantique. De plus, l'intrication a un comportement intéressant dans l'algorithme de Grover : elle augmente jusqu'à un point médian, puis diminue jusqu'à la fin de l'algorithme. Cette propriété pourrait être utilisée pour vérifier la validité d'une implémentation de l'algorithme de Grover par exemple. La QFT a un comportement d'intrication moins régulier, mais ce comportement avec les polynômes de Mermin a été comparé à l'hyperdéterminant de Cayley, montrant certaines similitudes, mais aussi quelques différences. Ce travail a ensuite été transposé avec Grâce Amouzou sur Qiskit afin d'évaluer si ces résultats pouvaient être transposés aux processeurs NISQ (*Noisy Intermediate-Scale Quantum*), ce qui a donné lieu à un article sur le sujet par G. Amouzou, J. Boffelli, H. Jaffali, K. Atchonouglo et F. Holweck [ABJ+20]. À l'heure actuelle, le bruit des processeurs quantiques d'IBM est encore trop important pour obtenir des résultats correspondant aux résultats simulés, mais l'exécution sur leur simulateur a donné les mêmes résultats que les nôtres. L'étude de la QFT a validé que certaines portes affecteraient l'intrication, et d'autres non, cette propriété des portes pourrait être utilisée dans la spécification et la validation.

Avec Jessy Colonval, nous nous sommes engagés ensuite dans le domaine des géométries finies en implémentant en Magma des algorithmes de génération de géométries préexistants [Cd19]. Ce travail était un échauffement pour la génération de géométries finies avec Magma, suivi par la génération de géométries finies dites quantiques pour leur lien avec les géométries de Mermin-Peres. Ces géométries ont été étudiées en détail : nous avons d'abord travaillé sur des familles de géométries, en évaluant si leurs membres étaient des géométries contextuelles ou non [dHGM21a, dHGM21b]. Nous avons découvert que, pour $n \in [2..5]$ qubits, les lignes de $W(2n-1,2)$ – l'espace symplectique des n qubits, un espace codant les relations de commutation dans le groupe de Pauli sur n qubits –, leurs restrictions aux hyperboliques et leurs restrictions aux ellipses sont toutes contextuelles, mais les générateurs de $W(2n-1,2)$ et les lignes de $W(2n-1,2)$ restreintes à certaines sous-géométries appelées perpsets ne sont jamais contextuelles. Ensuite, nous avons étudié la structure de $W(2n-1,2)$ en utilisant une signature nouvellement définie [SdHG21]. Cette étude est complétée par des correspondances trouvées entre les espaces symplectiques de différentes tailles, et a été réalisée spécifiquement pour des géométries utilisant des observables de 2 à 5 qubits. Ces études visent à mieux comprendre la contextualité afin de trouver des propriétés remarquables qui pourraient être utilisées pour spécifier la contextualité dans les programmes quantiques, mais elles ne sont que la première étape d'un long chemin.

Perspectives

Bon nombre des sujets abordés durant cette thèse mériteraient un suivi. L'évaluation de l'intrication dans l'algorithme de Grover et la QFT à l'aide des polynômes de Mermin pourrait mener à une preuve formelle de la croissance puis décroissance de l'intrication pendant l'algorithme de Grover. La QFT semble plus difficile à étudier à cet égard à cause de de son comportement irrégulier, mais l'algorithme de Grover est un bon candidat. Par ailleurs, depuis cette étude, l'algorithme de Grover a été implémenté et son exactitude quant à la recherche de l'élément recherché a été prouvée dans SQIRE. Ceci signifie que nous pourrions nous appuyer sur ce travail, et nous concentrer sur l'implémentation des polynômes de Mermin. Ceci pourrait également être abordé avec un autre logiciel de formalisation, tel que Why3 où l'automatisation pourrait aider à accélérer le processus.

Une autre direction prometteuse est l'étude des géométries quantiques contextuelles. En effet, il semble que l'espace contenant les géométries ait une structure très riche, qui pourrait être explorée davantage. Ces géométries représentent des expériences mettant en évidence la contextualité quantique, mais la contextualité peut aussi être mise en évidence de manière plus générale par des protocoles, comme ceux employés dans les jeux quantiques. Cela signifie que des programmes pourraient être étudiés sous cet aspect, nous permettant peut être un jour de les spécifier en utilisant la contextualité.

De plus, nos travaux relèvent plusieurs conjectures, validées seulement pour un petit nombre de qubits. C'est un endroit où les logiciels de preuve formelle nous permettraient de prouver ces propriétés pour tout nombre de qubits. Un exemple d'une telle conjecture serait le fait que toutes les lignes de $W(2n-1,2)$ forment des géométries contextuelles pour tout nombre n de qubits.

Introduction

Motivations

These days, quantum computing is very present in public communications. From the national initiatives (France, UK, USA, and more... [Fel21, Pre20, Smi18]), up to the interest of individual researchers, this interest comes from the huge promises of quantum computing [GKS+21], but also from the technical challenges which yields a rich field of work. But as for classical computing before, the quantum computing faces a particularly human problematic: how can we trust the program?

Nowadays, this question is predominant in some vital fields, where any error of the program could cost human lives, such as the aero-spacial, the nuclear reactors or the subways [CM14, MLL19, INT99, LSP+07]. But it tends to spread to all domains: for example, we would like to ensure that our bank's website is trustworthy, implying the need for verified JavaScript [BRN+20]. Since, if they are used in the future, quantum programs would have very wide fields of application, we would like to ensure from the beginning that they solve the problem they were conceived to solve. This is especially true since, for most people, quantum computing is not intuitive whatsoever, which implies more mistakes while conceiving a program, which implies a greater need for correctness, *i.e.* the property for a program to indeed perform the task it is intended to.

Problematic

In this context, the object of this thesis is to prepare the transfer of some methods from classical computing to quantum computing. We will decompose the title of this thesis – *Computational studies of quantum contextuality and non-locality properties towards their formal verification* – in its components so we can describe them one by one. Starting by the last element: the formal verification of a program running on a quantum system is for now a distant ideal. Indeed, due to the youth of operational quantum processors, we – as users – lack the practice on quantum languages that would greatly facilitate formal verification, as in the case of classical programs. Furthermore, the processes involved when designing quantum processors are yet hard to scale up, resulting in the creation of quantum program verification tools without computers able to run said quantum programs. Since reasoning on a program requires properties, such as being positive or even for an integer, we chose to study quantum specific properties in the eventuality that they could be of use for quantum program verification. In addition, these properties may be used to demonstrate that the implemented program could not be efficiently reproduced by a classical computer. This is where the choice of the properties is important, and this is why we chose contextuality and entanglement. Furthermore, this field being quite active, we were partially able to use existing tools – for classical and quantum verification – to study these properties. The idea being that we are preparing the field for quantum programs formal verification by studying properties that can later be used to specify quantum programs, in a similar way to quantum program verification preparing the field for scalable proved programs, once the technical limitations of building a quantum processor will be overcome.

It should be noted that, as stated previously, quantum computer scientists lack practice on writing code for quantum processors, but practice is not completely nonexistent either. In the recent years, some working quantum processors have been making the news [Qua, Por21, Mat21], but those processors are not yet capable of complex operations. Still, IBM gave control over some of their quantum processors to the people, through the *IBM quantum experience* [IBM]. This is a valuable tool to test small enough problems, and it has been used for this thesis, despite its important limitations.

Plan

The notions needed for the global understanding of the thesis are explained in Part I. This is followed by my contributions in Parts II to III. Some notions are only introduced when needed, when they are not relevant for the whole manuscript, to avoid throwing at the reader too much information at once.

Part I is split in two, with Chap. 1 introducing classical formal methods and Chap. 2 covering quantum computing, as seen by a computer scientist. This chapter covers the basics of quantum computing, the languages used in quantum computing, some notions of quantum program verification, and the two properties at the heart on this manuscript: entanglement, a manifestation of non-locality, and contextuality.

Part II covers a quantum circuit execution simulator I developed (Chap. 3) to study entanglement in Grover's algorithm (a search algorithm) and the Quantum Fourier Transform (QFT, a quantum spin on the Discrete Fourier Transform) (Chap. 4). In this part, the reader will also find the transposition of this study to Qiskit, IBM's library for quantum circuit simulation also providing the ability for the user to send the programs to be executed on their quantum processors, a form of quantum cloud computing (Chap. 5).

Finally, Part III is the junction of works on finite geometries interesting for their connection to quantum contextuality. The first work was the generation of finite geometries – also known as block designs – and checking that the generation algorithms were correct (Chap. 6). The second work (Chap. 7 and 8) was about the generation of finite geometries in order to study the symplectic space $W(2n-1,2)$, a finite vector space equipped with a function encoding the Pauli operators and their commutation relations.

List of publications

My Ph.D. works were published or are in the process of being published. For each publication, a short description is given below as well as the chapter in this thesis where it is tackled.

[Cd19], published (national workshop), Chap. 6: preliminary work to study algorithms generating block designs, mathematical objects in relation with quantum contextuality.

[dJH+20], poster (national conference), Chap. 4: study of entanglement in two famous quantum algorithms: the Grover algorithm and the Quantum Fourier Transform (QFT), the latter being a part of the Shor algorithm.

[dJH+21], published (*Q2* international journal), Chap. 4: article version of [dJH+20] about entanglement evaluation in Grover's algorithm and the QFT.

[dHGM21b], poster (international conference), Chap. 7: detection of contextuality proofs among subgeometries of binary symplectic polar spaces, with a computer.

[dHGM21a], in progress, Chap. 7: article version of [dHGM21b] about contextuality proofs among subgeometries of binary symplectic polar spaces.

[SdHG21], published (*Q1* international journal), Chap. 8: classification of polar subspaces of $W(2N-1,2)$ according to several characteristics, such as the number of their negative lines or the distribution of types of observables.

A website aggregating the pieces of code involved in each article was also created [dB21]. It is a hub for quantum verification code distribution for our team, and was also used to host other pages such as the 2019 GT IQ's days web page which we organized, GT IQ being CNRS's task force on quantum information, annually meeting to exchange on quantum information related works. In addition of helping organizing 2019's meeting, I presented there some of my work on Mermin's polynomials. It was a global will for me to exchange on quantum computing as broadly as possible: in this effort, in addition to some presentations directly related to my work, I presented quantum computing to *quantum-agnostic* people, such as the team of researchers of my lab, other Ph.D. students, and even motivated high school students.

I

Context

Table of Contents

This part aims at presenting some basic notions necessary for understanding the work accomplished during these last three years. Not every notion studied are presented here though, since some notions are needed only for the understanding of a restricted part of my work. Such notions are introduced, where they are needed, in order to avoid having an unreadable block of definitions, followed by some work far from the definitions, such as one may need to go back to the present part to recall the definitions. This approach though has the disadvantage that the definitions are not centralized, which results in the fact that they may be hard to find if need be. The solution to this problem is an Index on page 145. The present part presents an introduction to formal methods in Chap. 1 and an introduction to quantum computing in Chap. 2.

1

Formal methods

The aim of this thesis being to work toward quantum program verification, one needs to first establish a baseline for general program verification. A program can be said to be verified if some methods ensuring that it fulfills its objectives have been applied.

These methods can be classified in two broad categories: they can either be static or dynamic. Static analysis of the code does not run the code, and analyses it without execution, and dynamic, on the other hand, does run the code. Some methods of static analysis are deductive verification, where the code and its specifications are used to generate proof obligations; or abstract interpretation, where domains for the variables are considered instead of their exact values, in order to obtain information about the domain of the output.

These methods are complementary: because of their real-world implementation, even though the program may not have any undesired behavior in theory, it may behave unexpectedly when confronted to other surrounding systems. Since a single method does not suffice to definitively certify the correctness of a program, I consider some applications of three of the above mentioned methods: one static – deductive verification in Sec. 1.1 – and two dynamics – runtime assertion checking in Sec. 1.2 and program testing in Sec. 1.3.

1.1/ Deductive verification

When considering software engineering, some may only think about writing the intended software, but a task at least as important is to ensure that the software does what it is supposed to do. In mathematics, the only way to tell that an equation is valid is to prove it. Computer science has its roots in mathematics, as the Pascal's *Pascaline* [VPL19] for restricted calculations, the Babbage's *analytical engine* [Bro82] for more general calculation and Turing's *Bombe* [Gre14] for cryptography are testimonies. This embedding of computer science in mathematics from the very beginning gives us an insight of why programs can be seen as mathematical functions, operating on the memory of the computer. This gives us an intuition on the reason why proving properties about a program is even possible. There are several ways to achieve such proofs [Flo67, Hoa69, Dij75]. In Hoare logic (also called Floyd-Hoare logic), a program has a precondition (a logic formula on the inputs of the program), and a postcondition (a logic formula on the outputs of the program), the aim being to prove the postcondition if the precondition is met. Each atomic statement composing the program having its own pre and post condition (the core of the proof is to find them), the proof is complete when the prover (be it a human or a program) is able to stitch them together.

In order to prove properties of a program using Hoare logic, one presents all the needed information as a triple $\{P\}p\{Q\}$ where P is the precondition, p the program, and Q the post-condition. This way, reasoning about the program is quite straight forward, the program being decomposed as a sequence of operations, and each operation having some known effects on the logic. For example, combining sequentially two programs p and p' is done as follows:

$$\begin{cases} \{P\}p\{Q\} \\ \{Q\}p'\{R\} \end{cases} \implies \{P\}p;p'\{R\}.$$

1.2/ Runtime Assertion Checking

Unfortunately, it can be very hard to prove a program. This is why a software engineer has other tools on his belt to ensure that a software is doing what it is meant to do. Runtime Assertion Checking (RAC in short) is the process of evaluating various properties while the program is running. It ensures that the program is not misbehaving and sends a signal when it is. The mechanism of exceptions(in Python, Java, and more...) can be seen as an example of RAC. RAC is most often used in development to help the programmer to have a clear vision of the state of the memory, and assertions are generally dropped in production (once the program is executed by its end users). This means that RAC has a positive influence on the behavior of the code, but contrary to deductive verification, it does not provide any guarantee. This brings us to a very broad category of software development methods, which can use RAC but are not limited to it: testing.

1.3/ Testing

Testing in software design has a huge role in designing well behaving software. Testing is also a common practice in many other engineering domains. Software testing can be automated, to be as extensive as possible. Ideally, any situation presented to a software has been previously tested, and the behavior of the software is known ahead of time. But this is almost never possible as the number of combinations of the inputs of the software is very often considerable. For this reason, various methods of testing have been introduced to cover a spectrum of situations as wide as possible. Some examples of test method are unitary testing (where functions are tested independently of each other, as a form of divide and conquer strategy), model based testing (where a model of the world is theorized around the software to synthesize tests close to real world situations), analysis of tests coverage (to assess what parts of the code have been tested), ...

More information about software testing and RAC can be found in reference manuals, such as [UL07]. But beyond those specific practices, verification always focuses on properties. Most commonly, those properties are about correctness, speed or efficiency, but this rises the question of quantum properties: which one will eventually be central in quantum programs? A first lead would be about accessing which properties could be verified, such property should be decidable, but other conditions may need to be added.

2

Quantum information

Quantum physics was established at the beginning of the 20th century and progress in the understanding of its core concept – the quantization of energy – triggered revolutionary changes in the understanding of information flow in our universe. From the beginning, information flow was at the heart of the dilemmas about quantum physics, with for example Einstein speaking about *spooky actions at a distance*: quantum mechanics seemed to break the limit of the speed of light for the transfer of information, which is impossible in the current understanding of the physics of our universe. This very dilemma has since been solved, without requiring us to reconsider Einstein's relativity, but the questions about information flow in quantum mechanics have been the source for a new field, quantum computing: a way to manipulate quantum systems in order to perform computation out of reach for our "classical" computers.

These computations unavailable to our classical computer require the understanding of some elementary mechanics of quantum computing. The elements presented in this chapter will not cover all possible paradigms but they will give the reader a broad understanding of quantum computing. This understanding will be completed as needed in Parts II and III of this manuscript. Sec. 2.1 is a presentation of essential notions of quantum computing, Sec. 2.2 shows how quantum algorithms can be represented, both for human reading and machine usage, Sec. 2.3 gives a short state of the art for quantum verification, and Sec. 2.4 presents some quantum properties central for this thesis.

2.1/ Computation model

The most famous model of automated computation is the Turing machine [Pos36]. It is composed of a data storage, a program storage and a read-write head. At the very beginning of computer science, when the name of the field was not even established yet, it was shown that this machine could perform any arbitrary computation. Our computers are now only refined implementations of this Turing machine. The model used for quantum computing has notable similarities: a memory – qubits registers – and a program – a sequence of read-write operations. These elements will be presented in this section.

In all this thesis, we work on finite numbers of qubits. A qubit – or quantum bit – is the basic unit for quantum computation. A single qubit is most often represented as a two dimensional unit vector. In this vector space, the canonical basis is the set of vectors $\left\{\binom{1}{0}, \binom{0}{1}\right\}$. Those vectors are denoted $|0\rangle$ – read “ket zero” – and $|1\rangle$ – read “ket one” –, following Dirac’s “bra-ket” notation. They are the quantum equivalent to the classical bits and are called the computational basis. But since the space of states is a vector space, any other basis can be chosen to perform computations.

This vector representation does not allow for noisy systems. But since my work does not study them, we can consider qubits as vectors without any further loss of generality.

The laws of quantum mechanics dictate that the vector space in which we operate must be a complex Hilbert space, and that states in this space must be normalized. This lets us write a single qubit as $|\varphi\rangle = a\,|0\rangle + b\,|1\rangle = \binom{a}{b}$ with $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$. We can now express a state composed of several qubits. If the qubits are on the computational basis, we write them as $|0..0\rangle$, $|0..01\rangle$, $|0..10\rangle$,... $|1..1\rangle$. The numbers in the ket notation are the integers from 0 to $2^n - 1$ in binary basis, with n being the number of qubits, so we add some simplifying notations by declaring that the computational basis is the set of vectors $\{|0\rangle, ..., |N-1\rangle\}$ with $N = 2^n$ (there are 2^n combinations of 0’s and 1’s of length n). With this, a general vector of n qubits is written as $|\varphi\rangle = \sum_{i=0}^{N-1} a_i\,|i\rangle$ with $\sum_{i=0}^{N-1} |a_i|^2 = 1$. One can also represent two states side by side, for example if we have two registers[1] with the state $|\varphi_1\rangle$ on the first one and $|\varphi_2\rangle$ on the second one, then the state of the whole system will be defined as $|\varphi\rangle = |\varphi_1\rangle \otimes |\varphi_2\rangle$. The $\otimes$ operator is called the tensor product or the Kronecker product, and it has the following effect: if $|a_1\rangle$ and $|a_2\rangle$ are two base vectors with a_1 and a_2 the binary representation of some numbers, then $|a_1\rangle \otimes |a_2\rangle = |a_1 a_2\rangle$ where $a_1 a_2$ is the concatenation of the two binary representations. The effect of $\otimes$ on general vectors can be deduced by linearity of the tensor product.

Furthermore, the tensor product can be extended to the whole Hilbert space. Say we have the state $|\varphi\rangle$ in the Hilbert space $\mathcal{H}$, if $|\varphi\rangle = |\varphi_1\rangle \otimes |\varphi_2\rangle$ with $|\varphi_1\rangle$ from $\mathcal{H}_1$ and $|\varphi_2\rangle$ from $\mathcal{H}_2$, then we can write $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$. And we also use the power notation: for any objects, $O \otimes O \otimes \ldots \otimes O$ with n O’s can be written $O^{\otimes n}$.

The states are modified along the computation, most commonly through gates: like logic gates for classical computers, quantum gates have an effect on the qubits they operate on, which is dictated by their semantics, their formal meaning. The most common way to express the semantics of a gate is by the matrix having the following effect on the state: if

[1]A register is a figurative location for information to be stored. In our quantum computing context, we say that a register contains one or several qubits. The states can be seen as snapshots of the registers at a given time.

the state $|\varphi\rangle$ goes through a gate of matricial semantics M, then the state evolves to $|\varphi'\rangle = M|\varphi\rangle$. From this point on, when this is no cause for confusion, we will call M a gate which has the matricial semantics M. These gates must follow some mathematical restrictions. In particular, they must be reversible and their inverse is always their conjugate transpose, such matrices are said to be hermitian. Note that the conjugate transpose of an operator is called its adjoint and is noted $O^\dagger = \overline{O}^T$ where O^T is the transpose of O, *i.e.* the matrix such that the element of the i^{th} line and j^{th} column is $O^T_{(i,j)} = O_{(j,i)}$.

Let us consider some examples of gates. The most common gates are the Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} \text{ and } Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

They generate the Pauli group, containing the matrices $\{sI, sX, sY, sZ / s \in \{1, -1, i, -i\}\}$. These gates are in close relation with the computational basis, as well as other common bases such as $\{|+\rangle, |-\rangle\} = \left\{\frac{|0\rangle+|1\rangle}{\sqrt{2}}, \frac{|0\rangle-|1\rangle}{\sqrt{2}}\right\}$. The Hadamard gate H lets us switch between these bases, with the following semantics: $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$ ($|0\rangle \overset{H}{\leftrightarrow} |+\rangle$ and $|1\rangle \overset{H}{\leftrightarrow} |-\rangle$). It often the source of superposition in quantum algorithm.

Those are the most common one-qubit gates, but some multi qubit gates happen to be very common too. Firstly, a gate can be controlled: one or several qubits can make the gate behave as a wire if one of them is equal to $|0\rangle$ and as the controlled gate if they are all equal to $|1\rangle$. Any gate can be controlled in this manner, but $CNOT$ and the Toffoli gate are the most well known examples. $CNOT$ or Controlled-NOT or Controlled-X gives us another glimpse in the role of the Pauli X gate: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$, so X acts as the NOT gate for the computational basis, hence the $CNOT$ name. The matricial semantics of the CNOT gate is:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

It often has a role to play in the creation of entanglement, a quantum property presented in Sec. 2.4.1. Another useful gate is the Toffoli gate. It is a controlled $CNOT$ and its matricial semantics is:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Notice a pattern between $CNOT$ and T: the construction of the semantics matrix of a controlled gate is generalizable as follows. The control gate CG of an arbitrary gate G is built using the identity matrix I of same size as G as such: $CG = \begin{pmatrix} I & 0 \\ 0 & G \end{pmatrix}$.

A set of gates is said to be universal if it can transform any state of the Hilbert space into any other state of the Hilbert space using only transformations from the set of gates. There are several examples of universal sets of gates. A widely used one is given by the rotation gates – generalization of the Pauli gates –, the phase shift gate and $CNOT$. The matricial semantics of the rotation gates are $R_x(\theta) = \exp(-iX\theta/2) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$, $R_y(\theta) = \exp(-iY\theta/2) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$, $R_z(\theta) = \exp(-iZ\theta/2) = \begin{pmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{pmatrix}$. The matricial semantics of the phase shift gate is on the other hand $P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & \exp(i\theta) \end{pmatrix}$.

At this point, the states can be modified, but not returned to the user. In order to be able to read the content of a qubit register we use observables. An observable is a self-adjoint operator in a Hilbert space. It is used to encode the measure of a physical property of a given system. The Pauli gates are examples of such observables. An observable has a basis attached as it is diagonalizable: its basis is composed of its eigenvectors. Similarly, the eigenspaces are sometimes referred to as subspaces of the observable.

A common structure for observables can be given by a family of orthonormal vectors $(|\phi_i\rangle)_i$ (*i.e.* vectors such that $\langle\phi_i|\phi_j\rangle = \delta_i^j$ [2]), then $\sum_i \lambda_i |\phi_i\rangle\langle\phi_i|$ is an observable, with $\langle\phi_i|$ – called "bra ϕ_i" – the transpose conjugate of $|\phi_i\rangle$. In this case $(|\phi_i\rangle)_i$ is said to be the basis of the observable.

The possible values returned by a measure are the eigenvalues of the corresponding observable. But a measure has more impact than just returning the measured value to the user. Indeed, if the measured state is a basis state of the observable, then the measure will simply return the proper eigenvalue, but if the state is in a superposition of vectors of the basis, then the measure will be probabilistic.

In the general case, the measure projects the state on an eigenspace of the observable

[2] δ_i^j is the Kronecker operator: $\delta_i^j = 1 \iff i = j$ otherwise $\delta_i^j = 0$

with a probability equal to the norm of the projected state, then re-normalizes it, and returns the eigenvalue corresponding to this sub-space.

A more detailed explanation would be the following: take the observable O for the measure, let E_λ be the eigenspace associated with the eigenvalue λ, and note $meas(O, |\varphi\rangle)$ the measure operation, then the measure will have two effects, one on the state measured $|\varphi\rangle$, noted $\mapsto$, and the second will be the value returned, noted $\rightsquigarrow$. Let also $|\varphi\rangle_\lambda$ be the projection of $|\varphi\rangle$ on E_λ, $|\varphi\rangle_\lambda = Proj(|\varphi\rangle, E_\lambda)$, then

$$meas(O, |\varphi\rangle) : \begin{cases} |\varphi\rangle \mapsto \frac{|\varphi\rangle_\lambda}{\||\varphi\rangle_\lambda\|} \\ \rightsquigarrow \lambda \end{cases} \quad \text{with probability } \left\||\varphi\rangle_\lambda\right\|^2.$$

Because of normalization, it is reassuringly easy to check that the sum of all probabilities is equal to 1.

For example, the usual observable for the computational basis is $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. Its eigenvectors are $|0\rangle$ associated to 1 and $|1\rangle$ associated to -1. From this, we can deduce that the measure of $|\phi\rangle = a|0\rangle + b|1\rangle$ has a probability $|a|^2$ to return 1 and a probability $|b|^2$ to return -1 when it is measured by this observable.

The qubit formalism is the mathematical tool used to describe a physical experiment, like manipulating the spin of electrons, or the polarization of photons or the energy level of an atom. In each of these cases, the details of the implementation are quite different, and they can be found in the literature with different notations, for example instead of $|0\rangle$ and $|1\rangle$ the polarization of light can be denoted as $|\uparrow\rangle$ and $|\leftarrow\rangle$. The observable represents the measuring instrument used by the entity operating it (the physicist in a lab for example).

Finally, as stated previously, I only consider non-noisy states in this thesis, also called pure states. Pure states are in opposition with mixed states, described by density matrices instead of vectors.

2.2/ Algorithms and languages

A quantum algorithm is a series of statements involving quantum computing notions. There are several ways to formalize quantum algorithms, a very common one is to sequentially describe all the gates operating on the data. This formalism is called quantum circuitry and consists of wires representing the qubits, on which gates and measures operate.

Fig. 2.1 depicts such a circuit, and we can observe on this figure some key features of a circuit: it is generally written from left to right, it has an input, some transformations, and an output. For a given algorithm, some inputs may not be data to be worked on,

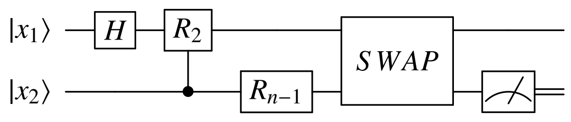


Figure 2.1: Example of quantum circuit

but simply additional wires to temporarily store data onto, and some output data can be discarded. This would be the equivalent of local variables in a classical program.

For many reasons, the use of circuits is very restrictive, both to work with and to think with. Here is a short list of shortcomings of quantum circuits:

- they do not allow loops;
- they are not a compact way to store a program;
- the gates involved generally do not allow for registry size change; which implies that each local variable must have its own space in the inputs and outputs.

But all these shortcomings have solutions by extending the language. This direction does not seem that promising though: since circuits were never meant to be a compact and efficient way to represent programs. They are mostly a pedagogic tool to visualize transformations.

Quantum circuits could be compared to assembly language, as it is a basic way of laying down the instruction for the processor to execute them. But even then, most assembly languages allow for subroutines, which is not the case for quantum circuit. This is why an early machine language for quantum programs was QASM [BI17] – quantum assembly –, a language to store basic quantum programs organized sequentially (with the addition of subroutines). This format is practical to transfer program from a computer to an other, but lacks the flexibility of modern languages. For this reason, Domain Specific Languages (DSL) compilable to QASM have been created in these modern languages to ease the creation of complex quantum programs. The python library Qiskit [Cro18] from IBM is an example of such a DSL: it allows its user to build quantum programs, and to run them either on a simulator or on their quantum processors available through internet. In addition to this library, IBM published a web interface to graphically create circuits and run them called Composer shown in Fig. 5.1. These efforts to make quantum computing easily available to beginners is a key point toward making more users at ease with the quantum computing paradigm. Through the rich interface of Qiskit, IBM's computer usage is not limited to beginners. We even used it for some of our experiments, presented in Chap. 5.

But since quantum computing has such a different underlying structure compared to classical computing, some have even wondered if operation sequences is the proper format for quantum programs. Diagrammatical languages such as ZX-calculus [Bac14] were cre-

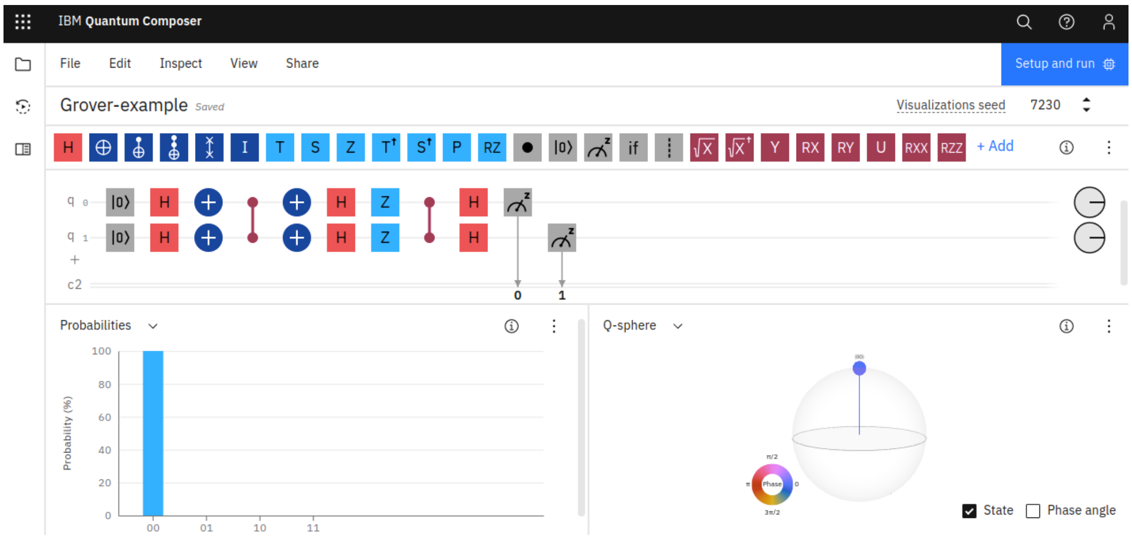


Figure 2.2: User interface for IBM's composer

ated as a potential solution to this problem. ZX-calculus works by abstracting the notion of gate as a notion of node, nodes are more elementary but have a much more powerful set of rewriting rules, allowing representing quite complex programs as fairly compact diagrams. It is still very much a work in progress, but it promises to enable powerful quantum compilers, provers, and more generally intermediate data manipulation tools (thanks to its rewriting rules) [Kv20]. ZX-calculus is not the ultimate representation for quantum programs though: because of its abstract nature, it is not easy to come up with new programs directly in ZX-calculus.

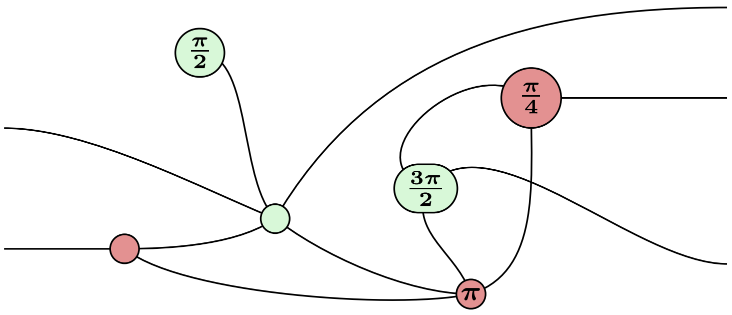


Figure 2.3: Example of ZX-diagram

Since circuitry is still the most common basis for quantum algorithms, we will use it in spite of its limitations.

2.3/ Quantum verification

In Sec. 1.1, I defined classical deductive verification: this notion was generalized to quantum computing. Since this thesis aims at producing tools to enable quantum verification, I explore in this section what quantum verification is at this time. Although the reader must keep in mind that this is a quickly evolving field, meaning that the picture depicted here may be quite different from the actual usage of quantum verification.

After learning about Hoare logic and in the path to transferring classical skills to quantum computing, one would logically have the idea of generalizing it as quantum Hoare logic [Yin18]. This work is quite recent and has a limited scope, but is nonetheless an interesting lead on the kind of methods used to reason about quantum programs. We did not follow this lead though because the idea seemed flawed at its root: the authors of this work replaced pre and postconditions by quantum predicates, which are in this case observables of the system. This usage of global observables is problematic because unrealistic: observables' sizes grow exponentially with the number of qubits. And even then, this is assuming than one considers a finite number of qubits: in this work, to simplify the thought process, the system is considered of infinite size, allowing the logic to ignore any size calculation.

Another promising project is Qbricks, published this year (2021). It uses Why3[3] to verify quantum algorithms [CBB+21]. This code was published too late for us to use it, but it allowed us to focus on specific quantum properties to study, in the goal of using them for specifying quantum problems. The idea behind this reasoning being that more information available for specification would enable easier choice of properties used for said specification. Why3 is a platform for deductive program verification providing a language for specification and programming, called WhyML, and relying on external theorem provers, both automated and interactive, to discharge verification conditions. Using Why3, Qbricks was able to automatically prove the correction of several quantum algorithms.

This last point is in fact what we chose to focus on: instead of directly tackle quantum verification, we studied properties that may come useful for specification of quantum programs.

2.4/ Quantum properties

As mentioned in Sec. 2.3, verifiable quantum properties could be keys to an efficient quantum verification process. In this context, we focused on two widespread and related properties: entanglement and contextuality. Those two properties are historically linked to differentiating quantum experiences from classical ones, but with the advancement in their understanding, they became more granular, which implies that these properties are not only destined to differentiate between quantum and classical situations, but can now differentiate between different *classes* of quantum states. These properties have also

[3]http://why3.lri.fr/

proved themselves to be core components of quantum programs and protocols (quantum teleportation, quantum telepathy games, ...).

2.4.1/ Entanglement

Entanglement is defined using separability. A state is separable if it can be decomposed as a tensor product of single qubits states. If a state is not separable, then it is entangled. In other words, a n-qubits state $|\varphi\rangle \in H^{\otimes n}$ is entangled *iff* there is no $(|\varphi_i\rangle)_{i\in[1..n]} \in H^n$ such that $|\varphi\rangle = |\varphi_1\rangle \otimes \ldots \otimes |\varphi_n\rangle$.

A famous example of entangled state is the Bell state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. This state is a great example of the strange repercussions of the quantum measure. Indeed, measuring in the computational basis either of the first qubit of this state would project the second qubit to be the same state as the one measured. At this point, the name "entanglement" for this phenomenon should make more sense: entangled particles have effect on one another, no matter what the distance is.

Another phenomenon is of importance: non-locality. It is a phenomenon where action on a part of the system has immediate effects on another part of the system, and it is enabled by entanglement. This is in fact the property that threw Einstein off when he first learnt about it, because it seemed to break the speed of light limit for transfer of information. But because of the limitations of quantum mechanics, this phenomenon in fact does not allow for information transfer faster than light [ER88].

Since the first days of quantum information, the process used to detect entanglement has been refined to return a continuum. I will call in this thesis degree of entanglement – or simply entanglement when there is no possible confusion – this continuous measure of entanglement.

One may think that the entanglement being a continuum, it would allow to discriminate between any two states. But this is in fact false. Indeed, due to how entanglement is defined, some operations on a state do not affect its degree of entanglement. This is formalized by the LU, LOCC and SLOCC classes. The term comes from set theory, where a class or equivalence class is defined for an equivalence relation $\mathcal{R}$ as a set of elements C such that $\forall a, b \in C, a\mathcal{R}b$.

LU stands for Local Unitary, and it is the group of unitary operators, *i.e.* operators O such that their adjoint is their inverse $OO^\dagger = I$, that acts separately on all qubits. Mathematically, an LU transformation is an operator O such that $\exists\,(O_i)_{i\in[1..n]} \in U_2$ s.t. $O = O_1 \otimes \ldots \otimes O_n$ with U_2 being the group of complex unitary matrices of size 2. With this, we can define LU-equivalent states as states equal to each other up to a LU transformation *i.e.* $|\varphi_1\rangle$ and $|\varphi_2\rangle$ are equivalent *iff* there exists a LU transformation O such that $|\varphi_1\rangle = O\,|\varphi_2\rangle$. LU is a natural group to consider since noiseless quantum operations are unitary: LU then only adds one restriction to this.

LOCC stands for Local Operations and Classical Communication, it is a broader set of operations than LU since it contains *single qubits unitary operations*, but also *measurements* and classical communication. An example of classical communication in quantum computing would be a gate applied only if a previous measurement had a given value. In the case of pure states though, LU and LOCC have the same equivalence classes [Vid00, Aul12].

Finally, SLOCC is the broadest class of the three. SLOCC stands for Stochastic Local Operations and Classical Communication, and this group is the group of *locally invertible operations*. Contrary to the previous two groups, SLOCC does not preserve entanglement: it cannot create entanglement from a separable state, but it can change the degree of entanglement of a state.

To recap, we have the following implications:

$$\text{LU-equivalence} \implies \text{LOCC-equivalence} \implies \text{SLOCC-equivalence.}$$

And in the case of pure states, on which I will focus in this manuscript, we even have

$$\text{LU-equivalence} \overset{pure}{\iff} \text{LOCC-equivalence.}$$

Several ways to evaluate entanglement of a state have come up along the years. For instance entanglement variations during the execution of Grover's algorithm have been studied either by computing the evolution of the Geometric Measure of Entanglement [RBM13, WG03], or by computing other measures of entanglement like the concurrence or measures based on invariants [BOF$^+$16, WG03, HJN16]. Among those measures of entanglement, only the last one is presented in this manuscript, in Appendix C defining the Cayley hyperdeterminant. Similarly, for Shor's algorithm and in particular to study the variation of entanglement within the QFT, numerical computation of the Geometric Measure of Entanglement was carried out in [SSB05]. Let us also mention [JH19] where the evolution of entanglement in Grover's and Shor's algorithms is studied qualitatively by considering the classes of entanglement reached during the execution of the algorithms.

The authors of [BOF$^+$16] proposed to exhibit the non-local behavior of the states generated by Grover's algorithm by testing a generalization of Bell's inequalities known as Mermin's inequalities, based on Mermin polynomials [ACG$^+$16, CGP$^+$02]. Let us define these polynomials that we will use in Part II.

Definition 2: Mermin polynomials, [ACG+16]

Let $a = \left(a_j\right)_{j\geq 1}$ and $a' = \left(a'_j\right)_{j\geq 1}$ be two families of one-qubit observables with eigenvalues in $\{-1, +1\}$. The Mermin polynomial $M_n(a, a')$ is inductively defined by:

$$\begin{cases} M_1(a, a') = a_1 \\ \forall n \geq 2,\ M_n(a, a') = \frac{1}{2} M_{n-1}(a, a') \otimes (a_n + a'_n) + \frac{1}{2} M_{n-1}(a', a) \otimes (a_n - a'_n) \end{cases} \tag{2.1}$$

Note that since we only use this definition for the Mermin polynomial, we will omit the (a, a') part of the notation when there is no ambiguity, to improve readability.

The reader familiar with the Mermin polynomial may also be surprised by the slightly different definition in comparison to the one introduced in [ACG+16]. In place of our $M_n(a', a)$, they had a second polynomial $M'_n(a, a')$ obtained by interchanging all the operators with and without the "′" mark of the definition. This is due to a simplification we introduced noticing that $M'_n(a, a') = M_n(a', a)$.

Example 1: *For $n = 2$, the Mermin polynomial is $M_2 = \dfrac{1}{2}(a_1 \otimes a_2 + a_1 \otimes a'_2 + a'_1 \otimes a_2 - a'_1 \otimes a'_2)$. When we chose $a_1 = Z, a_2 = (Z + X)/\sqrt{2}, a'_1 = X$ and $a'_2 = (Z - X)/\sqrt{2}$ the operator M_2 is, up to a factor, the CHSH operator used to prove Bell's Theorem [CHSH69].*

One can note that a one-qubit observable a with eigenvalues in $\{-1, +1\}$ can be written as a normed linear combination $a = \alpha X + \beta Y + \gamma Z$ of the Pauli matrices $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$ and $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, with the constraint $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$.

Mermin's inequalities

$$\langle M_n \rangle^{LR} \leq 1 \qquad \text{and} \qquad \langle M_n \rangle^{QM} \leq 2^{\frac{n-1}{2}} \tag{2.2}$$

respectively formalize that the expectation value $\langle M_n \rangle$ of M_n is bounded by 1 under the hypothesis LR of local realism, while it is bounded by $2^{\frac{n-1}{2}}$ in quantum mechanics (QM). The hypothesis of local realism (or principle of locality) states that an object is only affected by its immediate surrounding. $\langle M \rangle$ is the expected value of the observable M, i.e. a real number depending on the measured state. For the state $|\varphi\rangle$, the expected value of M is $\langle M \rangle_\varphi = \langle \varphi | M | \varphi \rangle$ and it represents the average of multiple measurements of $|\varphi\rangle$ by M. This value is here refined by adding a condition on $|\varphi\rangle$: $\langle M \rangle^{LR}$ means that $|\varphi\rangle$ follows the hypothesis of local realism, and $\langle M \rangle^{QM}$ means that $|\varphi\rangle$ follows the laws of quantum mechanics.

The violation of the first Mermin inequality shows non-locality which is only possible under the hypothesis of quantum mechanics and if the quantum state is entangled. More precisely the maximal violation of Mermin's inequalities occurs for GHZ-like states [Mer90, CGP+02, ACG+16], *i.e.* states equivalent to $|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n})$ by local transformations.

One of the advantages of Mermin's inequalities over other methods for evaluating entanglement, such as the hyperdeterminant, is that they can be tested by a physical experiment. Recently the violation of Mermin's inequalities was tested for $n \leq 5$ qubits on a small quantum computer [AL16, ABJ+20].

2.4.2/ Contextuality

While entanglement is most often approached as an evidence of non local behavior of quantum states, contextuality is rather about properties of compatible measurements in quantum physics and how these measurements satisfy some contextual properties. Since both of these phenomena are still under intensive study, their distinction is in fact quite fuzzy, some works have even pointed toward a common phenomenon at the root of those quantum properties [Mer93, AB11]. Historically though, entanglement and contextuality have been put into evidence from different considerations, and this difference resulted on different and complementary approached to tackle the studies of these properties. For this reason, I will present contextuality as a completely separate property despite the existing links with entanglement.

Quantum contextuality was first discovered with the work of Kochen and Specker. The Kochen-Specker Theorem [KS67] is a no-go result that states that any classical theory that would reproduce the outcomes of measurements in quantum physics has to be contextual. Here contextual means that if a classical theory predicts the outcomes of a measurement, then these outcomes should depend on the context, i.e. the set of compatible experiments that are performed before or after the measurement. In this manuscript we only consider operator-based proof of quantum contextuality as discovered by Mermin [Mer93].

$$\begin{tikzcd}
X\otimes I \arrow[r, dash] \arrow[d, dash] & I\otimes X \arrow[r, dash] \arrow[d, dash] & X\otimes X \arrow[d, equal] \\
I\otimes Y \arrow[r, dash] \arrow[d, dash] & Y\otimes I \arrow[r, dash] \arrow[d, dash] & Y\otimes Y \arrow[d, equal] \\
X\otimes Y \arrow[r, dash] & Y\otimes X \arrow[r, dash] & Z\otimes Z
\end{tikzcd}$$

Figure 2.4: The Mermin-Peres magic square.

Let us consider the configuration of two-qubit observables called the Mermin-Peres square represented in Fig. 2.4. Each vertex represents a two qubit observable consisting of measurement of each particle in the X, Y or Z directions or no measurement

(I). The lines are sets of mutually commuting observables, *i.e.* compatible measurement, they form a collection of six contexts. The outcome of each individual measurement is $+1$ or -1 as the Pauli matrices square to identity. However the product of the operators on each context is either $+I$ (simple line) or $-I$ (doubled line). A consequence of the laws of quantum physics is that the product of the measured eigenvalues of each observable of a given context has to be an eigenvalue of the product of the observables. Therefore the measurements of each context are subjected to constraints imposed by the sign of the context. While all these constraints are satisfied by the measurement of any context if we follow the rules of quantum mechanics, it is clear that there is no classical function that can assign $+1$ or -1 to each node of this square configuration and satisfy all the constraints. Indeed if we multiply all the lines of the Mermin-Peres square one should get $-I$ while if we consider the product of the outcomes of each node provided by the classical function one should get $(\pm 1)^2 \times \cdots \times (\pm 1)^2 = 1$ as each node belong to two contexts. This contradiction shows that, in order to satisfy the constraints, the classical function should be context-dependent, *i.e.* provide different results according to which context is measured. This argument furnishes a proof of the Kochen-Specker Theorem.

In chapters 7 and 8 we will study configuration of observables, like the Mermin-Peres square, that provide alternative proofs of the Kochen-Specker Theorem. We will call such configurations contextual.

II

Quantum Algorithms execution, simulation and entanglement studies

When considering code behavior, the first idea that comes in mind for checking it is to run it. This is the reason why I started building a simple simulator while starting my state of the art, to be able to easily play with the algorithms I crossed in my readings. Although many quantum circuit execution simulators already existed, creating my own one allowed me to tweak it to my needs, and this advantage had only a minimal cost, due to the low complexity of building a quantum circuit simulator for such needs. The created simulator and some examples of application are described in Chap. 3. The following work resulting in an article [dJH$^+$21] composes the Chap. 4. This second chapter, in addition to explaining Grover's algorithm and the QFT, presents our study of entanglement evolution in these algorithms. The specificity of this entanglement study is that our measure of entanglement is realizable on an actual quantum computer, which enabled me in Chap. 5 to rewrite my code to be able to use Qiskit to perform the same computation on IBM's publicly available quantum processor. All the code written for each of these chapters is freely available and documented on [dB21].

Table of Contents

3 A SIMULATOR FOR QUANTUM VERIFICATION

To start studying quantum verification, we first build a small quantum circuit simulator in Python, using a mathematics software system called SageMath, built on top of existing packages such as NumPy, SciPy and others... It was used to simulate Deutsch and Deutsch-Jozsa algorithms (resp in Sec. 3.2 and Sec. 3.3), and to check experimentally that they both operate as they are supposed to. In addition to the added control over the computation thanks to the homemade simulator, or maybe because of it, I was able to introduce exact computation (in opposition to floating point precision computation) of quantum states using cyclotomic fields. This approach was added as an option to be toggled, and event though it was not used in the published article because of its time execution cost (as implemented, it was more than 100 times slower than using floating point precision) it is promising because it is easier to formally reason on exact computation than on intervals. This could be useful for formal reasoning approaches. A few other additional benefits of creating an in-house simulator are presented along this chapter, such as the freedom of the input format, the ease of interoperability with other in-house software, and custom format for execution logs.

3.1/ The simulator

The simulator is extremely simple, a circuit is given to it as a list L of lists of matrices as well as a initial state V_0 and the simulator will return the list of states between the elements of L. Each element l of L represents a layer of the circuit and each element of l represents an operation carried on one or several wires. All the operations on a layer are considered as ran at the same time. For an example of input circuit, if `H` is the Hadamard matrix, `I2` and `I4` are the identity matrix (respectively in dimensions 2 and 4) and `X` is the first Pauli operator, then the circuit depicted in Figure 3.1 is represented by the list `[[H,I4], [X,X,I2], [I4,H], [H,H,H]]`. The identity is present in this formalism to represent the absence of operation at its position.

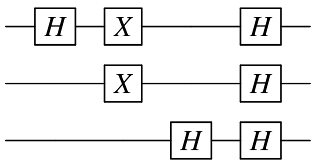


Figure 3.1: Circuit representation of `[[H,I4],[X,X,I2], [I4,H], [H,H,H]]`

The quantum measure presented in Sec. 2.1 can be simulated at any step of the circuit. For example, in Fig. 3.2, the output of the simulator would contain the five states $[V_0..V_4]$, and the measure would be simulated after the computation of V_3.

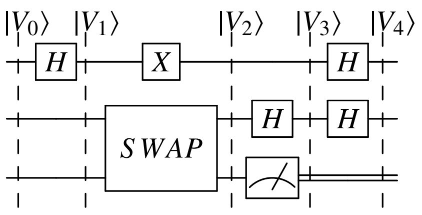


Figure 3.2: Example of mid circuit measure

The simulator's code is shown in listing 3.1. Note that the version here is somewhat a simplification in comparison to the version distributed on GitHub. This is done to make this manuscript more readable, but no core functionality has been altered by this simplification. This function takes as input the circuit and the initial state, computes for each layer the matrix corresponding to the whole layer, and then computes the state after the layer by multiplying the previous state by the layer matrix.

```python
def run(matrix_layers, V_init):
  V_running=cp.deepcopy(V_init)
  vectors_list=[V_running]
  matrices_list=[]
  for matrix_layer in matrix_layers:
    layerMatrix=Matrix([[1]])
    for running_matrix in matrix_layer:
      layerMatrix=kronecker(layerMatrix, running_matrix)
    V_running=layerMatrix*V_running
    matrices_list.append(layerMatrix)
    vectors_list.append(V_running)
  return vectors_list
```

Listing 3.1: Main function of the simulator

In fact, the complete version of this function has one more functionality, used only for experimentation: it also returns the matrix corresponding to each layer (the Kronecker product of all matrices of the layer). In addition to this, the `run` function has a display option: it can create LaTeX commands for each state and layer matrix as shown in listing 3.2 which is the output of the command `run(layers,v,output=True)`. This is done in order to ease presentation of quantum algorithms by automating them: if one uses this simulator to explain an algorithm, he will be able to add directly in his LaTeX presentation the steps of the algorithm using these commands. As the simulator is a work in progress though, there are obviously things to ameliorate. For example, implementing simplifications such as $1\,|1\rangle \to |1\rangle$ or $\frac{1}{2}\,\sqrt{2} \to \frac{1}{\sqrt{2}}$.

```
\newcommand{\Vzero}{\left(1,\,0,\,0,\,0,\,0,\,0,\,0,\,0\right)}
\newcommand{\Mone}{\left(\begin{array}{rrrrrrrr}
\frac{1}{2}\,\sqrt{2}&\frac{1}{2}\,\sqrt{2}&0&0&0&0&0&0\\
\frac{1}{2}\,\sqrt{2}&-\frac{1}{2}\,\sqrt{2}&0&0&0&0&0&0\\
0&0&\frac{1}{2}\,\sqrt{2}&\frac{1}{2}\,\sqrt{2}&0&0&0&0\\
0&0&\frac{1}{2}\,\sqrt{2}&-\frac{1}{2}\,\sqrt{2}&0&0&0&0\\
0&0&0&0&\frac{1}{2}\,\sqrt{2}&\frac{1}{2}\,\sqrt{2}&0&0\\
0&0&0&0&\frac{1}{2}\,\sqrt{2}&-\frac{1}{2}\,\sqrt{2}&0&0\\
0&0&0&0&0&0&\frac{1}{2}\,\sqrt{2}&\frac{1}{2}\,\sqrt{2}\\
0&0&0&0&0&0&\frac{1}{2}\,\sqrt{2}&-\frac{1}{2}\,\sqrt{2}
\end{array}\right)}
\newcommand{\Vone}{\frac{1}{2}\,\sqrt{2}\ket{000}+\frac{1}{2}\,
\sqrt{2}\ket{001}}
...
\newcommand{\Mthree}{\left(\begin{array}{rrrrrrrr}
1&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1
\end{array}\right)}
\newcommand{\Vthree}{\frac{1}{4}\,\sqrt{2}\ket{000}+\frac{1}{4}\,
\sqrt{2}\ket{001}+\frac{1}{4}\,\sqrt{2}\ket{010}+\frac{1}{4}\,
\sqrt{2}\ket{011}+\frac{1}{4}\,\sqrt{2}\ket{100}+\frac{1}{4}\,
\sqrt{2}\ket{101}+\frac{1}{4}\,\sqrt{2}\ket{110}+\frac{1}{4}\,
\sqrt{2}\ket{111}}
Usage: $\Vzero$ eventually becomes $\Vthree$.
```

Compiled result:
Usage: $\frac{1}{2}\,\sqrt{2}\,|000\rangle + \frac{1}{2}\,\sqrt{2}\,|001\rangle$ eventually becomes $\frac{1}{4}\,\sqrt{2}\,|000\rangle + \frac{1}{4}\,\sqrt{2}\,|001\rangle + \frac{1}{4}\,\sqrt{2}\,|010\rangle + \frac{1}{4}\,\sqrt{2}\,|011\rangle + \frac{1}{4}\,\sqrt{2}\,|100\rangle + \frac{1}{4}\,\sqrt{2}\,|101\rangle + \frac{1}{4}\,\sqrt{2}\,|110\rangle + \frac{1}{4}\,\sqrt{2}\,|111\rangle$.

Listing 3.2: Result of the LaTeX printing option of the simulator with a compiled usage example

Another display option is available as an auxiliary function: a function to display circuits. Examples of use are given in listing 3.3.

```
>>> circuit=[[('I',2),('H',1)],[('H',1),('H',1),('I',1)],[('I',1),('S',2)]]
>>> print_circuit(circuit, to_latex=False)
---H---
---H-S-
-H---|-
>>> print_circuit(circuit, to_latex=True)
\begin{align*}
 \Qcircuit @C=1em @R=.7em {
  & \qw       & \gate{H}  & \qw                & \qw\\
  & \qw       & \gate{H}  & \multigate{1}{S}   & \qw\\
  & \gate{H}  & \qw       & \ghost{S}          & \qw
 }
\end{align*}
```

Listing 3.3: Results of the printing function.

3.2/ Deutsch's algorithm

The first example on which I tried the simulator was Deutsch's algorithm [Deu85]. Let us see how I implemented it and how I verified it.

Deutsch's algorithm takes in a binary function $f : \{0, 1\} \to \{0, 1\}$ and returns in a single call of f whether the function is constant or not. The important part here is that the result is computed in a single evaluation of f, and it will later be developed into a slightly more impressive generalization.

To add some vocabulary, f is said to be a balanced function if $|f^{-1}(\{0\})| = |f^{-1}(\{1\})|$ where $f^{-1}(B)$ is the inverse image of the set B and $|A|$ is the cardinal of the set A. Given this, Deutsch's algorithm will return `true` if f is balanced and `false` if f is constant. In $\{0, 1\} \to \{0, 1\}$, f can only be one of those two.

To do so the function is encoded into its quantum equivalent, the oracle gate U_f built such as

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |f(x) \oplus y\rangle \,. \tag{3.1}$$

with $\oplus$ being the XOR operator.

The function creating the corresponding matrix is `U` defined in listing 3.4.

```
def U(f):
  result=Matrix(4, 4)
  for i in range(0, 4):
    result[i,(i//2)*2 + (i%2)^^(f[i//2])]=1
  return result
```

Listing 3.4: Function creating the oracle for Deutsch's algorithm

In this case, f is passed as a dictionary. For example, `f={0:0,1:0}` is the constant function returning 0.

The oracle building function requires some explanation: the image of each basis vector should be dictated by Eq. 3.1. In order to do this we decompose the index of the basis vector we intend to use: the i^{th} basis vector has its index decomposed as $i = 2i_1 + i_0$, $i_1 i_0$ is the binary representation of i, which means that i_1 corresponds to $|x\rangle$ and i_0 to $|y\rangle$ in Eq. 3.1. The image of this basis vector by the oracle will have its only non null entry in row $j = 2i_1 + i_0 \oplus f(i_1)$, this computation being the core of the content of the **for** loop in listing 3.4 (in SageMath, `^^` is the bitwise XOR operator). The binary representation of j is $i_1(i_0 \oplus f(i_1))$, which gives us the correspondence with Eq. 3.1.

The U_f gate is then used as shown in Fig.3.3.

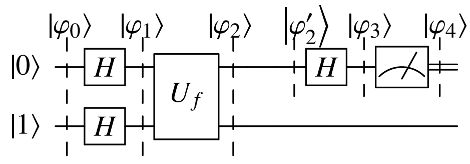


Figure 3.3: Deutsch's algorithm in circuit formalism

We recall that the matrix semantics of the gate H is equal to $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Let us develop the calculus to understand Deutsch's algorithm. For this calculus, each step φ_i corresponds to a number i of gates layer passed, and the layers are $H \otimes H$, U_f and $H \otimes I$:

$$\varphi_0 = |0\rangle \otimes |1\rangle$$
$$\varphi_1 = \frac{1}{2}(|0\rangle + |1\rangle) \otimes (|0\rangle - |1\rangle)$$
$$= \frac{1}{2}(|0\rangle \otimes (|0\rangle - |1\rangle) + |1\rangle \otimes (|0\rangle - |1\rangle))$$
$$\varphi_2 = \frac{1}{2}(|0\rangle \otimes (|f(0) \oplus 0\rangle - |f(0) \oplus 1\rangle) + |1\rangle \otimes (|f(1) \oplus 0\rangle - |f(1) \oplus 1\rangle)$$

At this point, if $f(0) = 0$, $|f(0) \oplus 0\rangle - |f(0) \oplus 1\rangle = |0\rangle - |1\rangle$, and otherwise it is the opposite. The idea is the same for $f(1)$.

$$= \frac{1}{2}\left((-1)^{f(0)} |0\rangle \otimes (|0\rangle - |1\rangle) + (-1)^{f(1)} |1\rangle \otimes (|0\rangle - |1\rangle)\right)$$
$$= \frac{(-1)^{f(0)}}{2}\left(|0\rangle + (-1)^{f(0)\oplus f(1)} |1\rangle\right) \otimes (|0\rangle - |1\rangle)$$

At this point we can ignore the last qubit, because it does not change anymore, and the global phase (complex multiplier of norm 1), because it has no effect on the measure.

$$\begin{aligned}\varphi_2' &= \frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^{f(0)\oplus f(1)}\,|1\rangle\right)\\ \varphi_3 &= \frac{1}{2\sqrt{2}}\left(|0\rangle + |1\rangle + (-1)^{f(0)\oplus f(1)}(|0\rangle - |1\rangle)\right)\\ &= \frac{1}{2}\left(\left(1 + (-1)^{f(0)\oplus f(1)}\right)|0\rangle + \left(1 - (-1)^{f(0)\oplus f(1)}\right)|1\rangle\right)\end{aligned}$$

With this, if f is constant then $(1+(-1)^{f(0)\oplus f(1)}) = 2$ and $(1-(-1)^{f(0)\oplus f(1)}) = 0$, which implies that the measure will have the following effect: $\varphi_4 \to |0\rangle$, and if f is not constant, we have the opposite case and $\varphi_4 \to |1\rangle$. For the reader not used to quantum computing, this is a nice example that shows that, although the measure is probabilistic, a quantum algorithm can be deterministic.

3.3/ A generalization of Deutsch's algorithm, Deutsch-Jozsa algorithm

An interesting generalization of Deutsch's algorithm, as stated before, is Deutsch-Jozsa algorithm [DJ92], which, this time, checks if the function is balanced or constant (it is a precondition of the algorithm for f to be one or the other). It operates on boolean functions with n inputs. (Note that if we do not restrict the function to be either constant or balanced, this algorithm becomes semi-deterministic, returning 1 for any constant function, and 0 with a non-null probability for all other functions.)

The formula defining U_f in this case is again Eq. 3.1, and it is built by the functions showed in listing 3.5. The proof that the oracle has this shape is shown in Appendix A.

```
def U(f):
  X=matrix([[0, 1], [1, 0]])
  I=matrix.identity(2)
  result=matrix(2*len(f))
  for i in range(0, len(f)):
    if f[i]==0:
      result[2*i:2*i+2,2*i:2*i+2]=I
    else:
      result[2*i:2*i+2,2*i:2*i+2]=X
  return result
```

Listing 3.5: Function creating the oracle for Deutsch-Jozsa algorithm

The circuit representing Deutsch-Jozsa algorithm is depicted in Fig. 3.4.

To understand the calculus implied in Deutsch-Jozsa's algorithm, two points should be clarified. Firstly, let us recall that $|a_1 \dots a_k\rangle$ stands for $|a_1\rangle \otimes \dots \otimes |a_k\rangle$. We take advantage of

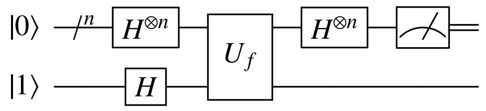


Figure 3.4: Deutsch-Jozsa algorithm in circuit formalism

this to represent arbitrary combinations of binary digits as their decimal representation: $|5\rangle = |5_2\rangle = |101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle$. Secondly, when fed $|0\rangle^{\otimes n}$, the Hadamard gate array is creating a superposition of all states: $H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle$, and for an arbitrary state $|\varphi\rangle = \sum_{k=1}^{2^n-1} \alpha_k |k\rangle$, its Hadamard transform would be $H^{\otimes n}|\varphi\rangle = \sum_{k=1}^{2^n-1} \alpha_k \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{k\cdot j} |j\rangle \right)$ with $k \cdot j = k_1 j_1 \oplus \ldots \oplus k_n j_n$ being the bitwise product of k and j.

Using the same tricks as in Deutsch's algorithm proof, we have the following calculus to prove Deutsch-Jozsa's algorithm:

$$\begin{aligned}
\varphi_0 &= |0\rangle^{\otimes n} \otimes |1\rangle \\
\varphi_1 &= \frac{1}{\sqrt{2^{n+1}}} \sum_{k=0}^{2^n-1} |k\rangle (|0\rangle - |1\rangle) \\
\varphi_2 &= \frac{1}{\sqrt{2^{n+1}}} \sum_{k=0}^{2^n-1} |k\rangle (|f(k)\rangle - |1 \oplus f(k)\rangle) \\
&= \frac{1}{\sqrt{2^{n+1}}} \sum_{k=0}^{2^n-1} (-1)^{f(k)} |k\rangle (|0\rangle - |1\rangle) \\
\varphi_2' &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{f(k)} |k\rangle \\
\varphi_3 &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{f(k)} \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} (-1)^{k\cdot j} |j\rangle \right) \\
&= \frac{1}{2^n} \sum_{j=0}^{2^n-1} \left(\sum_{k=0}^{2^n-1} (-1)^{f(k)} (-1)^{k\cdot j} \right) |j\rangle
\end{aligned}$$

Given this, the probability of measuring $|0\rangle^{\otimes n}$ is $\left| \frac{1}{2^n} \sum_{k=0}^{2^n-1} (-1)^{f(k)} \right|^2$ which is 1 if f is constant and 0 if f is balanced. This result is validated for system sizes up to 4 qubits by the code available on GitHub. It should be noted that these calculations were performed using the so called exact calculation of SageMath, using cyclotomic fields [Bir67] simply by adding the snippet of code shown in listing 3.6 and by replacing $\sqrt{2}$ by the newly defined `sqrt2` in the code. We went no further than 4 qubits because the code at this point was still a proof of concept.

```
field=UniversalCyclotomicField()
e8=field.gen(8)
sqrt2=e8 + conjugate(e8)
```

Listing 3.6: Snippet of code enabling exact calculation

3.4/ Conclusion

In these experiments, the results were quite trivial (since both of those algorithms are themselves quite trivial), but this proof of concept showed us that it was possible to use this simulator on harder problems. This is exactly what we did when using it to study Grover's algorithm and the Quantum Fourier Transform (QFT) with Mermin polynomials.

4

Mermin polynomials for entanglement detection in Grover's algorithm and Quantum Fourier Transform

This chapter reports on the content of [dJH+20, dJH+21], with some notions previously defined in this manuscript not repeated here.

4.1/ Introduction

Quantum entanglement has been identified as a key ingredient in the speed-up of quantum algorithms [JL03], when compared to their classical counterparts. Our work is in line with previous works on a deeper understanding of the role of entanglement in this speed-up [EJ98, BP02, CBAK13, KM06].

We focus on Grover's algorithm [Gro96] and the Quantum Fourier Transform (QFT) [NC10, Chap. II-Sec. 5] which plays a key role in Shor's algorithm [Sho94]. We choose these two examples because they both provide quantum speed-up (quadratic for Grover's algorithm and exponential for the QFT) and are well understood and described in the literature [NC10]. Previous work tackled entanglement in Grover's algorithm and the QFT from two perspectives: quantitatively, with the Geometric Measure of Entanglement (GME) [Shi95, BNO02, WG03], separately for Grover's algorithm [RBM13] and the QFT [SSB05], and qualitatively, by observing the different entanglement SLOCC classes traversed by an execution, for both algorithms [JH19].

Instead of directly measuring entanglement we use Mermin polynomials [Mer90, ACG+16, AL16] defined in Sec. 2.4 to demonstrate the non-locality (breaking of an upper bound holding for all classical states) of some states generated by these algorithms. Knowing that a state exhibits non-local properties allows us to conclude that the state is entangled. In this respect one uses Mermin polynomials as entanglement witnesses as suggested in [TG05, GT09]. Batle et al. [BOF$^+$16] previously investigated non-local properties during Grover's algorithm using Mermin polynomials. However they concluded to the absence of non-locality. In the present work we setup the Mermin polynomials in such a way that we exhibit, on the contrary, violation of the classical inequalities in Grover's algorithm. Moreover our evaluation techniques are more efficient, allowing us to reach 12 qubits. We also exhibit non-locality during the QFT in the context of Shor's algorithm.

An initial motivation of this study is the verification of quantum programs. Turning a quantum algorithm into an implementation for a quantum computer with scarce resources often requires highly non-trivial optimizations, which may introduce bugs in the resulting programs. Checking state properties is a way to gain more confidence in these implementations. In this chapter we investigate non-locality as a property of entangled quantum states that could be checked for a quantum algorithm and its implementations. In this respect evaluation of Mermin polynomials is of particular interest: violation of the classical bound has a physical meaning and the evaluation of Mermin polynomials can be implemented on a quantum computer, as it was demonstrated by Alsina et al. [AL16].

In this chapter we make two different uses of Mermin polynomials. In our study of Grover's algorithm we build for each number of qubits a specific Mermin polynomial which achieves maximal violation for the quantum state of highest GME that Grover's algorithm is meant to approach during its execution. Doing so we will not only show that the states generated by the algorithm violate the classical bound but also that the valuations of this specific Mermin polynomial behave similarly to the GME. In our study of the QFT, we propose a different approach by choosing at each step of the algorithm a Mermin polynomial whose valuation is maximal for the given state. We show that this quantity is a local unitary invariant that can be compared to other invariants. In the context of Shor's algorithm for four qubits, we also obtain violation of the Bell-like Mermin inequalities (also called MABK in the literature) during the QFT part of the algorithm. This amount of violation is not constant during the QFT, which shows a qualitative change of the nature of entanglement involved. This differs from the quantitative results obtained with the Groverian's measure of entanglement [SSB05] for which it was proved that the amount of entanglement is nearly constant in Shor's algorithm during the QFT. Without being contradictory the present work illustrates the fact that non-equivalent classes of entanglement under local unitary transformations are achieved during the QFT part of Shor's algorithm, as it was shown in [JH19].

The chapter is organized as follows. After Section 4.2 presenting some background on Grover's algorithm, the QFT and Mermin polynomials, Section 4.3 presents our method and results concerning the detection of entanglement in Grover's algorithm and the QFT. In particular we exhibit Mermin inequalities violations in both algorithms. In this section we also compare the results obtained with the Mermin polynomials to previous results [JH19] using the Cayley hyperdeterminant. Finally, Section 4.4 documents the code developed for this evaluation, in order to make it reusable by anyone wishing to[1]. In order to ease the reading of this chapter, we have gathered the more technical parts in the appendix of this manuscript. Firstly regarding a description of the states in Grover's algorithm in Chap. B and secondly the definition of the Cayley hyperdeterminant in Chap. C.

4.2/ Background

We recall here some notions. This manuscript relies on pure state formalism: each considered state is a normalized vector of the Hilbert space $\mathcal{H} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2$, this will be for us the definition of an Hilbert space. A separable state $|\varphi\rangle$ is a rank-one tensor, i.e., $|\varphi\rangle = |\varphi_1\rangle \otimes |\varphi_2\rangle \otimes \cdots \otimes |\varphi_k\rangle$, where $|\varphi_i\rangle$ are single-qubit states. A tensor/state $|\varphi\rangle$ is said to be of rank r if there are r rank-one tensors $|\varphi_i\rangle = \left|\varphi_1^i\right\rangle \otimes \left|\varphi_2^i\right\rangle \otimes \cdots \otimes \left|\varphi_k^i\right\rangle$, with $i = 1, \ldots, r$, such that $|\varphi\rangle = \sum_{i=1}^r \alpha_i |\varphi_i\rangle$ with $\alpha_i \in \mathbb{C}$, and r is minimal for this property. An entangled state is a tensor of rank higher than 1.

The remainder of this section provides necessary background to the reader, regarding Grover's algorithm (4.2.1), some properties of the states during its execution (4.2.2) and the Quantum Fourier Transform (4.2.3).

4.2.1/ Grover's algorithm

We summarize here Grover's algorithm, widely described in the literature ([Gro96, LMP03] and [NC10, chapter 6]).

Grover's algorithm aims to find objects satisfying a given condition in an unsorted database of 2^n objects, *i.e.* to solve the following problem.

Given a positive integer n, $N = 2^n$, $\Omega = \{0, \ldots, N-1\}$ and the characteristic function $f : \Omega \to \{0, 1\}$ of some subset S of Ω ($f(x) = 1$ iff $x \in S$), find in Ω an element of S only by applying f to some elements of Ω.

Grover's algorithm provides a quadratic speedup over its classical counterparts. Indeed, assuming that each application of f is done in one step, it runs in $O(\sqrt{N})$ instead of $O(N)$.

Figure 4.1 shows this algorithm as a circuit composed of several gates that we now describe. $H^{\otimes n+1}$ is simply the Hadamard gate on each wire. When applied on the first n

[1]The source code is available at https://quantcert.github.io/Mermin-eval .

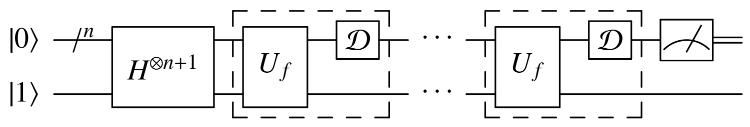


Figure 4.1: Grover's algorithm in circuit formalism

registers initialized at $|0\rangle$, it computes the superposition of all states, *i.e.*,

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{N}}\sum_{\mathbf{x}=0}^{N-1}|\mathbf{x}\rangle\,.$$

After $H^{\otimes n+1}$, the dashed box (hereafter called $\mathcal{L}$) is repeated $k_{opt} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{|S|}}\right\rfloor$ times.

The circuit $\mathcal{L}$ is composed of the oracle U_f and the diffusion operator $\mathcal{D}$. The gate U_f computes the classical function f. It has the following effect on states:

$$\forall(\mathbf{x},y)\in\{0,\dots,N\}\times\{0,1\},\ U_f\left(|\mathbf{x}\rangle\otimes|y\rangle\right) = |\mathbf{x}\rangle\otimes|y\oplus f(\mathbf{x})\rangle\,.$$

On the circuit of Figure 4.1 one can show that the last register remains unchanged when applying the U_f gate. Indeed after the Hadamard gate H, this last register becomes $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Now consider a state $|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Then

$$U_f\left(|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) = \begin{cases}|\mathbf{x}\rangle\otimes\frac{|1\rangle-|0\rangle}{\sqrt{2}} & \text{if } f(\mathbf{x})=1\\ |\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}} & \text{otherwise.}\end{cases}$$

In other words,

$$U_f\left(|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) = (-1)^{f(\mathbf{x})}\left(|\mathbf{x}\rangle\otimes\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right).$$

One says that the oracle U_f marks the solutions of the problem by changing their phase to -1. To emphasize this, we adopt the usual convention which consists of ignoring the last register and considering that U_f has the following effect:

$$\begin{cases}U_f\ |\mathbf{x}\rangle = -|\mathbf{x}\rangle\,, \forall\mathbf{x}\in S\\ U_f\ |\mathbf{x}\rangle = |\mathbf{x}\rangle\,, \forall\mathbf{x}\notin S\end{cases}.$$

The diffusion operator $\mathcal{D} = 2(|+\rangle\langle+|)^{\otimes n} - I_{2^n}$ performs the inversion about the mean. Indeed if $|\varphi\rangle = \sum_{\mathbf{i}=0}^{N-1}\alpha_{\mathbf{i}}|\mathbf{i}\rangle$ and $\bar{\alpha} = \frac{1}{N}\sum_{\mathbf{i}=0}^{N-1}\alpha_{\mathbf{i}}$ denotes the mean value of the amplitudes of $|\varphi\rangle$, then $\mathcal{D}\ |\varphi\rangle = \sum_{\mathbf{i}=0}^{N-1}\alpha'_{\mathbf{i}}|\mathbf{i}\rangle$ with $\alpha'_{\mathbf{i}} - \bar{\alpha} = \bar{\alpha} - \alpha_{\mathbf{i}}$.

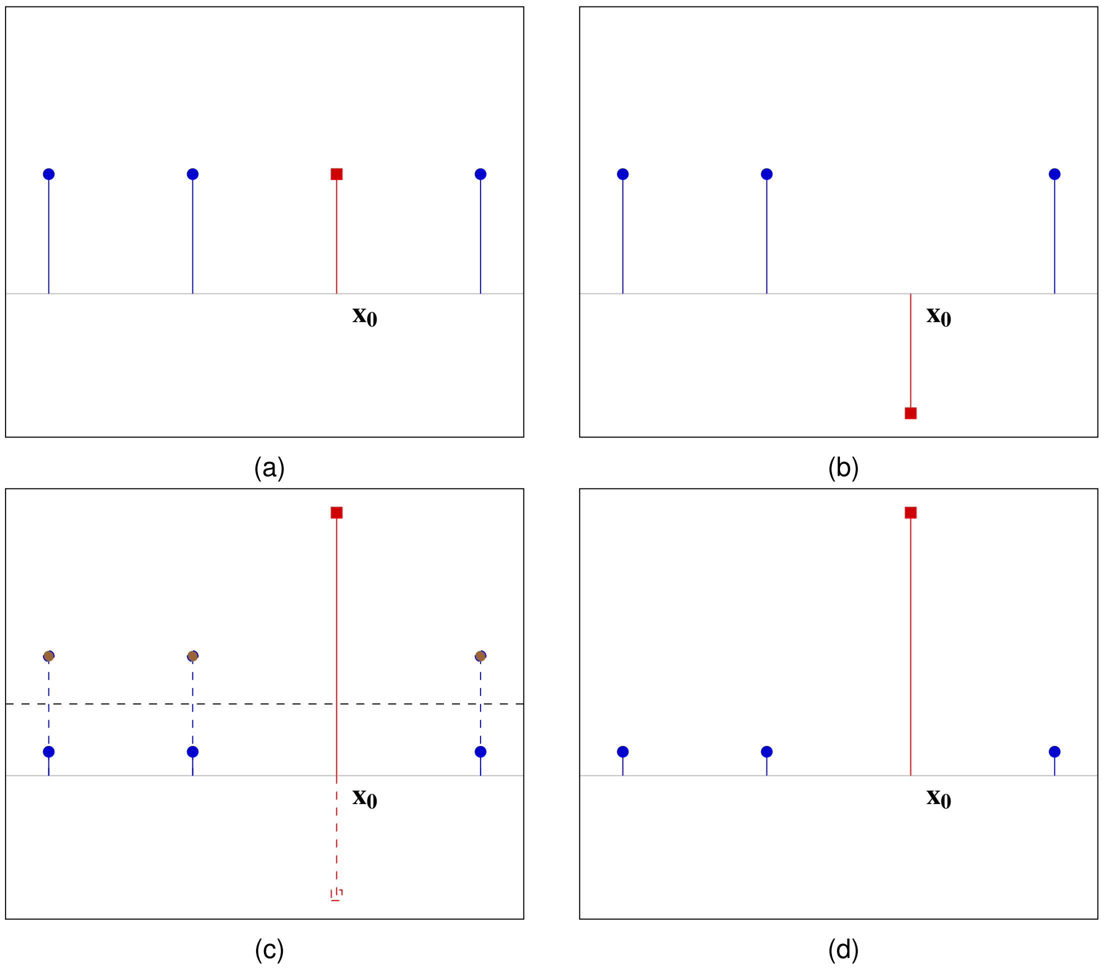


Figure 4.2: First iteration of loop $\mathcal{L}$ in Grover's algorithm

Figure 4.2 provides a visualization of the effect of the beginning of the algorithm on the amplitudes of $|\varphi\rangle$. For readability purposes, only 4 amplitudes are represented, and only one element is searched ($S = \{\mathbf{x_0}\}$), shown with a square instead of a bullet. The combs represent the amplitude of each element. The state is initialized to $|\mathbf{0}\rangle$. The state resulting of applying $H^{\otimes n}$ is the superposition of all states $|+\rangle^{\otimes n}$ (Figure 4.2a). Then the oracle U_f flips the searched element (Figure 4.2b), and the diffusion operator $\mathcal{D}$ performs the inversion about the mean (Figures 4.2c and 4.2d).

The final measure yields the index of an element from S with high probability.

4.2.2/ Properties of states in Grover's algorithm

The evolution of the amplitudes of the state $|\varphi\rangle$ during the execution of the algorithm is well known [NC10]. If we denote by θ the real number such that $\sin(\theta/2) = \sqrt{|S|/N}$, then after k iterations (*i.e.*, after applying k times the circuit $\mathcal{L}$), the state is:

$$|\varphi_k\rangle = \alpha_k \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \beta_k \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle \tag{4.1}$$

with $\alpha_k = \frac{1}{\sqrt{|S|}} \sin\left(\frac{2k+1}{2}\theta\right)$ and $\beta_k = \frac{1}{\sqrt{N-|S|}} \cos\left(\frac{2k+1}{2}\theta\right)$. The sequences $(\alpha_k)_k$ and $(\beta_k)_k$ are two real sequences respectively increasing and decreasing when k varies between 0 and $k_{opt} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{|S|}} \right\rfloor$.

An alternative representation of the evolution of the states during the execution of Grover's algorithm is proposed in [HJN16]. An elementary algebra calculation (See Appendix B, Proposition 4) shows that

$$|\varphi_k\rangle = \tilde{\alpha}_k \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \tilde{\beta}_k |+\rangle^{\otimes n} \tag{4.2}$$

with $\tilde{\alpha}_k = \alpha_k - \beta_k$ and $\tilde{\beta}_k = 2^{n/2}\beta_k$. The sequences $(\tilde{\alpha}_k)$ and $(\tilde{\beta}_k)$ are respectively increasing and decreasing on $\{0, \ldots, k_{opt}\}$ (see Appendix B, Proposition 5).

In particular, if one considers the case of one searched element $|\mathbf{x_0}\rangle$, *i.e.* $S = \{\mathbf{x_0}\}$, then Equation (4.2) becomes

$$|\varphi_k\rangle = \tilde{\alpha}_k |\mathbf{x_0}\rangle + \tilde{\beta}_k |+\rangle^{\otimes n}. \tag{4.3}$$

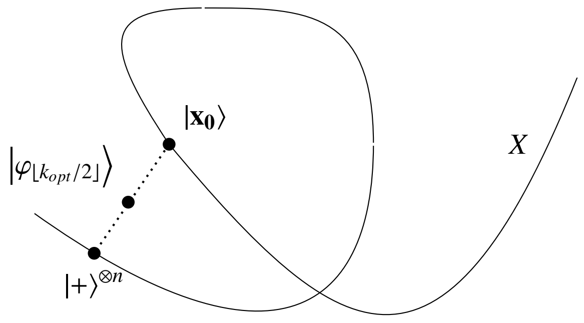


Figure 4.3: States in Grover's algorithm and the separable states [HJN16, Figure 2]

Figure 4.3 provides a graphical interpretation of Equation (4.3). The "curve" X represents the variety (set defined by algebraic equations) of separable states. This figure illustrates the fact that during the execution of Grover's algorithm, the quantum state $|\varphi_k\rangle$ evolves as follows: it starts from the separable state $|+\rangle^{\otimes n}$ and moves on the dotted secant line until it gets close to the searched state $|x_0\rangle$ when $k = k_{opt}$. All states on the secant line are entangled (rank-two tensors).

In [HJN16], it is proven that for states in superposition $\alpha|\mathbf{x_0}\rangle + \beta|+\rangle^{\otimes n}$ with $\alpha, \beta \in \mathbb{R}_+$, the GME is maximal when $\alpha = \beta$. Let $|\varphi_{ent}\rangle$ hereafter denote the state $(|\mathbf{x_0}\rangle + |+\rangle^{\otimes n})/K$, normalized with the factor K. Figure 4.3 suggests that the search should come close to the state $|\varphi_{ent}\rangle$, around the step $k_{opt}/2$. Thus, a maximum of entanglement is expected close to this pivot step.

4.2.3/ Quantum Fourier Transform (QFT)

The quantum analogous of the Discrete Fourier Transform (DFT) is the Quantum Fourier Transform (QFT). It acts linearly on quantum registers and is a key step in Shor's algorithm, permitting to reveal the period of the function defining the factorization problem [Sho94, NC10].

In the context of Shor's algorithm, the QFT is used to transform a periodic state into another one to obtain its period. The periodic state $\left|\varphi^{l,r}\right\rangle$ of n qubits with shift l and period r is defined by

$$\left|\varphi^{l,r}\right\rangle = \frac{1}{\sqrt{A}}\sum_{i=0}^{A-1}|l+ir\rangle \qquad \text{with } A = \left\lceil\frac{N-l}{r}\right\rceil \text{ and } N = 2^n,$$

for $0 \leq l \leq N-1$ and $1 \leq r \leq N-l-1$ [SSB05, Eq. 5].

For example, for the periodic 4-qubit states, with shift $l = 1$ and period $r = 5$, there are $A = \left\lceil\frac{16-1}{5}\right\rceil = 3$ basis elements, so:

$$\left|\varphi^{1,5}\right\rangle = \frac{1}{\sqrt{3}}(\,|\mathbf{1}\rangle + |\mathbf{6}\rangle + |\mathbf{11}\rangle\,) = \frac{1}{\sqrt{3}}(\,|0001\rangle + |0110\rangle + |1011\rangle\,)\,.$$

When applied to one of the computational basis states $|k\rangle \in \{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ (expressed here in decimal notation), the result of the QFT can be expressed by

$$QFT\ |k\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}\omega^{kj}\,|j\rangle\,,$$

where $\omega = e^{\frac{2i\pi}{N}}$ is the primitive N-th root of unity. Then, for any n-qubit state $|\psi\rangle = \sum_{j=0}^{N-1} x_j\,|j\rangle$, we get

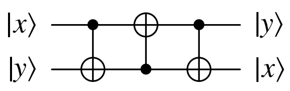
$$QFT\ |\psi\rangle = \sum_{k=0}^{N-1} y_k\,|k\rangle \quad \text{with } y_k = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} x_j\cdot\omega^{kj}. \tag{4.4}$$

The corresponding matrix is

$$QFT_N = \frac{1}{\sqrt{N}}\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}.$$

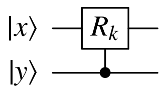
In the circuit representation, the QFT can be decomposed into several one-qubit or two-qubit operators. To obtain this decomposition three different kinds of gates are used: the Hadamard gate, the SWAP gate and the *controlled-R_k* gates, defined by the matrices and circuits

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



and

$$cR_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2i\pi}{2^k}} \end{pmatrix}$$



The complete circuit of the QFT is provided in Figure 4.4, where the n-qubit SWAP operation consists of swapping $|x_1\rangle$ with $|x_n\rangle$, $|x_2\rangle$ with $|x_{n-1}\rangle$, and so on.

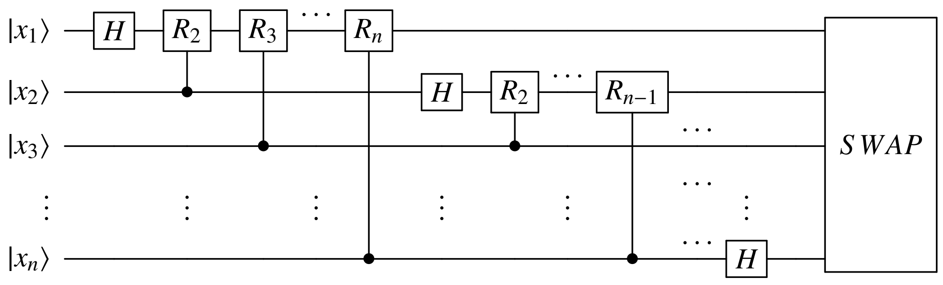


Figure 4.4: Circuit representation of the Quantum Fourier Transform for a n-qubit register

Remark 1: *One of the reasons that explain the exponential speed-up in Shor's quantum algorithm, is the complexity of the QFT which is quadratic with respect to the number of registers. By comparison, classically, the complexity of the Fast Fourier Transform algorithm that computes the DFT of a vector with 2^n entries is in $O(n2^n)$.*

4.3/ Method and results

In this section we present the main results of this study, obtained by evaluating Mermin polynomials on states generated at different steps of Grover's algorithm and the QFT. As explained in the introduction (see Sec. 4.1) our goal is to exhibit quantum properties of those states that can be experimentally checked. When it violates the classical bound, a Mermin polynomial detects entanglement – a resource that has been proved several times to appear in those algorithms. We obtain those violations in both algorithms. It is also known that the amount of violation of Mermin's inequalities is not in one-to-one correspondence with the quantity of entanglement involved [BGS05]. The question of measuring the quantity of entanglement is also a difficult question, as it is known that the notion of absolutely maximally entangled states does not exist already in the four-qubit case [HS00]. Here we compare evaluation of Mermin polynomials to different types of entanglement measures. In Grover's algorithm one uses a specific Mermin polynomial, which is fixed once for all the algorithm. By carefully choosing this polynomial one shows that its evaluation behaves like the GME. In the QFT algorithm, previous work [SSB05] concluded to small variations of the GME. Here, by choosing differently which Mermin polynomial we evaluate at each state, we show that the entanglement classes change during the QFT, as it was already observed in [JH19].

Once two families $(a_j)_{1\leq j\leq n}$ and $(a'_j)_{1\leq j\leq n}$ of observables are chosen, one can define the Mermin test function f_{M_n} by $f_{M_n}(\varphi) = \langle\varphi|M_n|\varphi\rangle$. Inequalities (2.2) tell that $f_{M_n}(\varphi) > 1$ implies that $|\varphi\rangle$ is non-local. We present in this section two approaches to choose the parameters $(a_j)_{1\leq j\leq n}$ and $(a'_j)_{1\leq j\leq n}$ of M_n to satisfy the previous inequality for some states generated by the quantum algorithm of choice.

The first approach evaluates each state that the algorithm goes through with the same function f_{M_n}, with a unique polynomial M_n chosen prior to state computation. This approach has the advantage of providing a fast calculation ($(a_j)_{1\leq j\leq n}$ and $(a'_j)_{1\leq j\leq n}$ are computed only once), but the function f_{M_n} is not a measure of entanglement, since it is not invariant by local unitary transformations, *i.e.*, we do not have $f_{M_n}(\varphi) = f_{M_n}(g.\varphi)$ for all transformations $g \in LU = U_2(\mathbb{C})^n$ and all quantum states $|\varphi\rangle$ ($|g.\varphi\rangle$ is defined as such: for $g = (g_1, \dots, g_n)$ and $G = g_1 \otimes \dots \otimes g_n$, $|g.\varphi\rangle = G\ |\varphi\rangle$).

The second approach is to choose a different M_n for each state $|\varphi\rangle$, by optimizing $f_{M_n}(\varphi)$ for each state traversed by the algorithm. This means that we are finding values for (a_j) and (a'_j) many times for a single run. This approach was for example used in [BOF+16]. We use it in Section 4.3.2.1 to define a quantity $\mu(\varphi)$, invariant under the group LU of local unitary transformations (see Proposition 1).

4.3.1/ Grover's algorithm properties

Hereafter we simplify the calculations by taking $S = \{\mathbf{x_0}\}$, *i.e.*, by considering that Grover's algorithm is only searching for a single element $\mathbf{x_0}$. We want to show two properties:

1. Grover's algorithm exhibits non-locality.
2. Parameters of the Mermin test function can be computed so that the function values increase and then decrease for the successive states $|\varphi_k\rangle$ in Grover's algorithm. The

maximum is reached at an integer k_{max} in $\{\lfloor k_{opt}/2 \rfloor, \lceil k_{opt}/2 \rceil\}$.

Property **1** is in contradiction with [BOF+16], a detailed explanation is given in Remark 2. Property **2** makes the chosen Mermin test function behave like the Geometric Measure of Entanglement.

Next section details the method we followed for finding a good Mermin polynomial establishing these properties.

4.3.1.1/ Method

The definition of Mermin polynomials provides degrees of freedom in the choice of $(a_j)_{j\geq 1}$ and $(a'_j)_{j\geq 1}$ (an infinite number of parameters). We reduce that choice by imposing that the two sequences $(a_j)_{j\geq 1}$ and $(a'_j)_{j\geq 1}$ are constant, *i.e.* $\forall j, a_j = a$ and $a'_j = a'$. This restriction strongly reduces calculations, and it will be sufficient to achieve our objectives.

Let us denote by a and a' the two one-qubit observables that will be used to write our Mermin polynomial. We have $a = \alpha X + \beta Y + \gamma Z$ and $a' = \alpha' X + \beta' Y + \gamma' Z$ with the constraints $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$ and $|\alpha'|^2 + |\beta'|^2 + |\gamma'|^2 = 1$.

The degrees of freedom are the 6 complex numbers α, β, δ, α', β' and δ' with the two normalization constraints. Let $A = (\alpha, \beta, \delta, \alpha', \beta', \delta')$ be the six-tuple of these variables.

In order to satisfy Property **2**, we search for a six-tuple of parameters A such that f_{M_n} reaches its maximum for the state $\varphi_{k_{opt}/2}$. We also would like this choice of A to be independent of the states generated by the algorithm. According to the geometric interpretation presented in Section 4.2.2, the state $\varphi_{k_{opt}/2}$ should tend to the state $|\varphi_{ent}\rangle = \frac{1}{K}(|\mathbf{x_0}\rangle + |+\rangle^{\otimes n})$ when n tends to infinity (the approximation improves as n increases). Moreover the state $|\varphi_{ent}\rangle$ is a tensor of rank two with an overlap $\langle \mathbf{x_0}|+\rangle^{\otimes n} = 1/\sqrt{2^n}$ between the states $|\mathbf{x_0}\rangle$ and $|+\rangle^{\otimes n}$ which tends to 0 as n increases, *i.e.*, we expect the state $|\varphi_{ent}\rangle$ to behave like a GHZ-like state when n is large (by definition the GHZ state is SLOCC equivalent to any non-biseparable rank-two tensor). This point is important because GHZ-like states are the ones that maximize the violation of classical inequalities by Mermin polynomials [Mer90, CGP+02, ACG+16]. Therefore by choosing a tuple of parameters A maximizing $f_{M_n}(\varphi_{ent})$ we expect to satisfy Properties **1.** and **2.**.

We use a random walk in $\mathbb{R}^6$ to maximize $f_{M_n}(\varphi_{ent})$. We operate the walk for a fixed number of steps, starting from an arbitrary point. At each step, we choose a random direction, and move toward it to a new point. If the value of $f_{M_n}(\varphi_{ent})$ at that new point is higher than at the previous one, then that point is the start point for the next step, otherwise a new point is chosen.

Once the proper coefficient for M_n found, we compute the values of each $f_{M_n}(\varphi_k)$ for k in

$\{0, \ldots, k_{opt}\}$ to validate Properties **1.** and **2.**.

Example 2: *When searching the state $|0000\rangle$, the highest value of $f_{M_4}(\varphi_{ent})$ obtained by this random walk is for $A = (-0.7, -0.3, -0.7, -0.5, 0.7, -0.5)$. Then, A is used to compute M_4, and then $f_{M_4}(\varphi_k), \forall k \in \{0, \ldots, k_{opt}\}$.*

Remark 2: *Some comments are in order at this point to compare our approach with the work of [BOF+16]. First in [BOF+16] all calculations are done using the density matrices formalism instead of the vector/tensor approach we use here. But this difference is meaningless, because we are only considering pure states, so, every computation in the density matrix formalism can be done equivalently within the vector state formalism. Moreover in [BOF+16] the optimization is done at each step of the algorithm with respect to the state computed by the algorithm, while we compute the parameters only once with respect to a targeted state $|\varphi_{ent}\rangle$. Finally, as mentioned at the beginning of Section 4.3.1.1, we also restrict ourselves to two operators a and a' and thus all optimizations are performed on six parameters instead of $6n$. This allows us to perform the calculation for a larger number of qubits (up to 12).*

4.3.1.2/ Results

Thanks to our implementation of this method in SageMath, described in Section 4.4, we obtain the values depicted in Figure 4.5, for n from 4 up to 12 qubits. The searched element $\mathbf{x_0}$ is always the first element $|\mathbf{0}\rangle$ of the canonical basis, but other searched elements would give similar results, by symmetry of the problem.

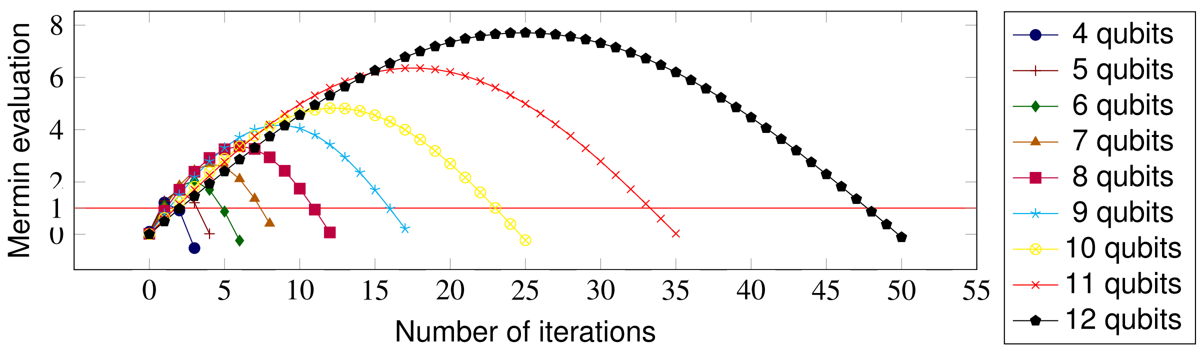


Figure 4.5: Mermin evaluation during Grover's algorithm for $4 \leq n \leq 12$ qubits

The lower bound for the number n of qubits is set to 4 because for $n \leq 3$ the algorithm has no time to show any advantage, is not very reliable and doesn't exhibit non-locality. The upper bound is set to 12 because of technological limitations: computations for 13 qubits or more become too expensive.

We see that the two expected properties hold for all values of n: the classical limit is violated and the Mermin evaluation increases up to the middle of the executions, and then decreases (the maximal values are given in Figure 4.6).

n	4	5	6	7	8	9	10	11	12
$\lfloor k_{opt}/2 \rfloor$	1	1	2	3	5	8	12	17	24
k_{max}	1	2	3	4	6	9	12	18	25
$f_{M_n}(\varphi_{k_{max}})$	1.21	1.72	2.05	2.69	3.37	4.17	4.83	6.36	7.71

Figure 4.6: Maximums of $f_{M_n}(\varphi_k)$ for $4 \leq n \leq 12$ qubits

Remark 3: *In [BOF+16] similar curves (Figure 3) were obtained for $n \in \{2,4,6,8\}$ qubits showing the increasing-decreasing behavior, but the violation of Mermin's inequalities – the non-locality – was not established for $n = 6$ and $n = 8$, whereas it is obtained in our calculation. Recall from Remark 2 that the calculation of [BOF+16] is not exactly the same as the one performed in this chapter. The curves of [BOF+16] are obtained by maximizing $f_{M_n}(\varphi_k)$ at each step of the algorithm with a larger number of parameters. Therefore as we obtain violation of Mermin's inequalities via a restricted calculation, the authors of [BOF+16] should also have observed it. We suspect errors in the implementation of the calculation of Equations (19) of [BOF+16] as we have redone this calculation for $n = 6$ based on Equations (18) and (20) of [BOF+16], and we have obtained the violation of Mermin's inequalities shown in Figure 4.7.*

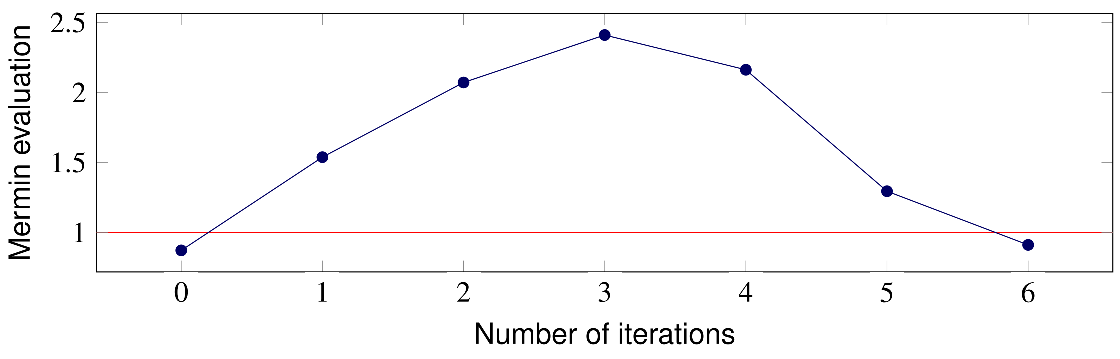


Figure 4.7: Mermin evaluation during Grover's algorithm for 6 qubits using [BOF+16] method

Remark 4: *The curve for $n = 12$ in Figure 4.5 should be compared to the curve of Figure 1 of [RBM13] where the evolution of the GME of the states generated by Grover's algorithm is given for $n = 12$ qubits. In our setting it is not a surprise that both curves are similar because in all of our calculations the function f_{M_n} is defined by the set of parameters that maximizes its value for $|\varphi_{ent}\rangle$. Similar behavior for other invariants in the context of Grover's algorithm have also been observed in [MW02a, CBAK13, HJN16].*

Figure 4.8 provides another argument explaining why we expected violation of Mermin's inequalities in Grover's algorithm when n increases. The curve with points as dots corresponds to the evaluation of $f_{M_n}(\varphi_{ent})$ and the curve with points as crosses corresponds to the theoretical upper bound for the violation of the Mermin inequality defined by M_n. It can be deduced from the geometric description of the algorithm (Section 4.2.2) that the quantum state $\left|\varphi_{\lceil k_{opt/2}\rceil}\right\rangle$ should be close to $|\varphi_{ent}\rangle$ and thus behave like it with respect to the Mermin polynomial. Despite the fact that $f_{M_n}(\varphi_{ent})$ does not reach the theoretical upper bound that is obtained for states LOCC equivalent to $|GHZ_n\rangle$, one sees that the difference between $f_{M_n}(\varphi_{ent})$ and the classical bound 1 increases as a function of n.

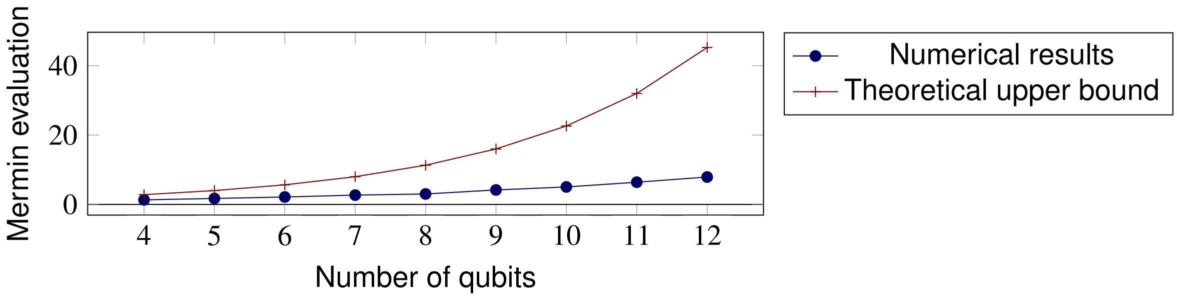


Figure 4.8: Numerical results compared to theoretical boundary of the Mermin evaluation

4.3.2/ Quantum Fourier Transform

To exhibit non-local behavior of states generated at each step of the Quantum Fourier Transform we restrict ourselves to periodic four-qubit states for the following reasons:

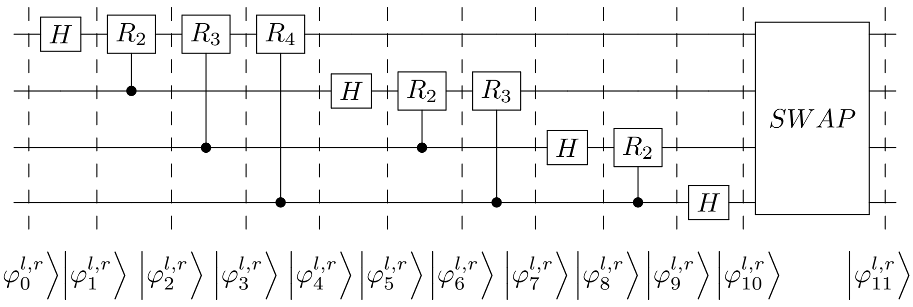
1. as explained in Section 4.2.3, the QFT in Shor's algorithm is applied to periodic states [NC10];
2. as we will see in Section 4.3.2.2 the four-qubit case is sufficient to obtain violation of Mermin's inequalities;
3. we want to compare the present approach with a recent study of entanglement in Shor's algorithm in the four-qubit case, proposed by H. Jaffali and F. Holweck [JH19].

4.3.2.1/ Method

When we apply the QFT to periodic states we have no *a priori* geometric information about the type of states that will be generated. In fact it depends on two initial parameters that define the periodic state $\left|\varphi^{l,r}\right\rangle$: its shift l and its period r. Therefore there are no reasons for restricting the choice of parameters in the calculation of $f_{M_n}(\varphi^{l,r})$. For the four-qubit case this implies that our optimization will be carried over the 24 parameters defining M_4, hereafter denoted $\alpha_1, \ldots, \alpha_{24}$ (such that $a_1 = \alpha_1 X + \alpha_2 Y + \alpha_3 Z, \ldots, a_4 =$

$\alpha_{10}X+\alpha_{11}Y+\alpha_{12}Z$, $a'_1=\alpha_{13}X+\alpha_{14}Y+\alpha_{15}Z$, ..., and $a'_4=\alpha_{22}X+\alpha_{23}Y+\alpha_{24}Z$), and this, for each state generated, in opposition to Section 4.3.1.

For $k\geq 0$, let $\left|\varphi_k^{l,r}\right\rangle$ denote the state reached after the first k gates in the QFT (Figure 4.9) initialized with the periodic state $\left|\varphi^{l,r}\right\rangle$ with shift l and period r (this means that $\left|\varphi_0^{l,r}\right\rangle=\left|\varphi^{l,r}\right\rangle$).



H
R2
R3
R4
H
R2
R3
H
R2
H
SWAP
$\left|\varphi_0^{l,r}\right\rangle$ $\left|\varphi_1^{l,r}\right\rangle$ $\left|\varphi_2^{l,r}\right\rangle$ $\left|\varphi_3^{l,r}\right\rangle$ $\left|\varphi_4^{l,r}\right\rangle$ $\left|\varphi_5^{l,r}\right\rangle$ $\left|\varphi_6^{l,r}\right\rangle$ $\left|\varphi_7^{l,r}\right\rangle$ $\left|\varphi_8^{l,r}\right\rangle$ $\left|\varphi_9^{l,r}\right\rangle$ $\left|\varphi_{10}^{l,r}\right\rangle$ $\left|\varphi_{11}^{l,r}\right\rangle$

Figure 4.9: Circuit representation of the QFT for 4 qubits with its numbered states

We are interested by the evolution of the function q defined for $k\geq 0$ by

$$q(k)=\max_{\alpha_1,\ldots,\alpha_{24}} f_{M_4}\left(\varphi_k^{l,r}\right). \tag{4.5}$$

In [JH19] two of the authors of the paper corresponding to this chapter have studied the evolution of entanglement for periodic four-qubit states through QFT by computing the absolute value of an algebraic invariant called the Cayley hyperdeterminant and denoted by Δ_{2222}. This polynomial of degree 24 in 16 variables is a well-known invariant in quantum information theory and its absolute value is known to be a measure of entanglement [MW02b, LT03, OS06, GW14]. We provide the definition of Δ_{2222} in Appendix C.

Surprisingly, the two approaches, which are of different natures – an algebraic definition for the hyperdeterminant and an operator-based construction for Mermin evaluation – would sometimes present similar behavior (see Figure 4.10).

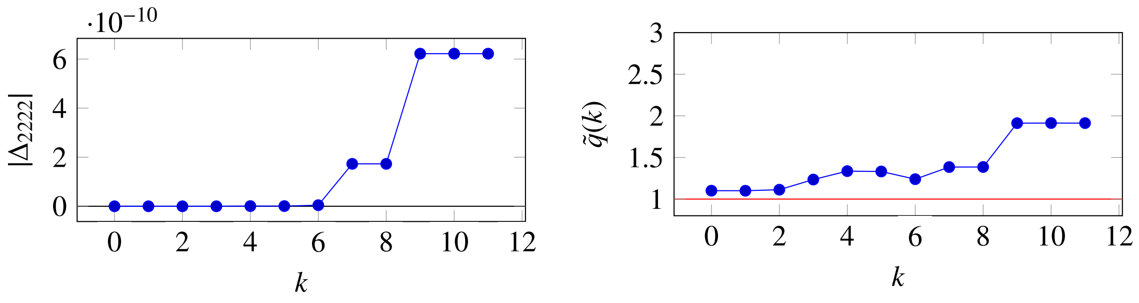


Figure 4.10: Evaluation of entanglement during the QFT of $\left|\varphi^{9,1}\right\rangle$ using the hyperdeterminant and Mermin evaluation

In [JH19] it was observed that the evolution of entanglement for four-qubit periodic states through QFT shows three different behaviors with respect to Δ_{2222}.

- Case 1. The polynomial Δ_{2222} is nonzero when evaluated on $\left|\varphi^{l,r}\right\rangle$ and does not vanish during the transformation. In terms of four-qubit classification [VDDMV02] it means that the transformed states remain in the so-called G_{abcd} class. This happens for $(l, r) \in \{(1, 3), (2, 3)\}$.
- Case 2. The polynomial Δ_{2222} is zero for the periodic state $\left|\varphi^{l,r}\right\rangle$ and is nonzero during the QFT. This happens for $(l, r) \in \{(0, 3), (0, 5), (2, 1), (3, 1), (3, 3), (4, 1), (4, 3), (5, 1), (5, 3), (6, 1), (6, 3), (7, 1), (9, 1), (10, 1), (11, 1), (12, 1)\}$.
- Case 3. The polynomial Δ_{2222} is zero for the periodic state $\left|\varphi^{l,r}\right\rangle$ and it remains equal to zero all along the QFT for all the other (l, r) configurations (with $0 \leq l \leq N - 1$ and $1 \leq r \leq N - l - 1$).

Before presenting the results let us point out that now the calculated quantity is invariant under local unitary transformations, *i.e.* under the group $LU = U_2(\mathbb{C})^n$.

Proposition 1: *Let $|\varphi\rangle \in (\mathbb{C}^2)^{\otimes n}$ be a n-qubit state. We recall that the Mermin polynomial M_n is a function of families of observables, as defined in Definition 2. With this, we define the function μ as*

$$\mu(\varphi) = \max_{a,a'} \langle\varphi|M_n(a, a')|\varphi\rangle . \tag{4.6}$$

Then $\mu(\varphi)$ is LU-invariant.

Proof. First one recalls that a one-qubit observable A such that $Sp(A) = \{-1, 1\}$ can always be written as $A = \alpha X+\beta Y+\gamma Z$ with $\alpha,\beta, \gamma \in \mathbb{R}$ and $\alpha^2 +\beta^2 +\gamma^2 = 1$. For the action $g.A = g^\dagger Ag$ on A by conjugation with a unitary matrix $g \in U_2(\mathbb{C})$, one has $g.A = \tilde{A} = \tilde{\alpha}X + \tilde{\beta}Y + \tilde{\gamma}Z$ with $\tilde{\alpha},\tilde{\beta}, \tilde{\gamma}$ reals such that $\tilde{\alpha}^2 + \tilde{\beta}^2 + \tilde{\gamma}^2 = 1$. Indeed $\tilde{A}$ is also a one-qubit observable such that $Sp(\tilde{A}) = \{-1, 1\}$.

Let us denote by $\lambda = (\alpha_1,\beta_1, \gamma_1, \alpha'_1,\beta'_1, \gamma'_1, \dots, \alpha_n,\beta_n, \gamma_n, \alpha'_n,\beta'_n, \gamma'_n) \in \mathbb{R}^{6n}$ a tuple of $6n$ real parameters such that $\alpha_i^2 +\beta_i^2 +\gamma_i^2 = 1$ and $\alpha'^2_i +\beta'^2_i +\gamma'^2_i = 1$ that define a Mermin polynomial $M_n(\lambda)$. Then

$$\mu(\varphi) = \max_{\lambda} \langle\varphi|M_n(\lambda)|\varphi\rangle$$

exists, because $\lambda \to \langle\varphi|M_n(\lambda)|\varphi\rangle$ is a continuous function on a dense compact interval. Let us denote by λ' a tuple of parameters that maximizes $\langle\varphi|M_n(\lambda)|\varphi\rangle$, *i.e.*,

$$\mu(\varphi) = \langle\varphi|M_n(\lambda')|\varphi\rangle .$$

Let $|\psi\rangle$ be a n-qubit state LU-equivalent to $|\varphi\rangle$. Thus there exists $g = (g_1, \dots, g_n) \in LU$ such that $|\psi\rangle = |g.\varphi\rangle = G|\varphi\rangle$ with $G = g_1\otimes\ldots\otimes g_n$. Then $\langle\varphi|M_n(\lambda')|\varphi\rangle = (\langle\varphi| G^\dagger)GM_n(\lambda')G^\dagger(G|\varphi\rangle) = \langle\psi|M_n(\lambda'')|\psi\rangle$ for some tuple of parameters λ''. To prove this, we first observe that $M_n(\lambda)$ is a sum of tensor products of normed linear sums of Pauli matrices. Since $G \in LU$, $GM_n(\lambda')G^\dagger$ is a tensor product of one qubit observables with a specter of $\{-1, 1\}$, which means that each of these observables can be expressed as a normed linear sum of Pauli matrices which which gives us λ'' such that $GM_n(\lambda')G^\dagger = M_n(\lambda'')$. Therefore

$$\mu(\varphi) \leq \mu(\psi).$$

But $|\varphi\rangle = G^\dagger |\psi\rangle$ also holds, so a similar reasoning provides the inequality $\mu(\varphi) \geq \mu(\psi)$ and thus the equality.

In the next section we plot and analyze different curves of the approximation $\tilde{q}$ of q in the four-qubit case for different choices of (l, r).

4.3.2.2/ Results

In order to compute the values of the function q defined by (4.5), we find an approximation noted $\tilde{q}$ by optimizing the parameters $\alpha_1, \ldots, \alpha_{24}$ defining M_n. Curves of $\tilde{q}(k)$ are shown on Figures 4.11, 4.12 and 4.13, for $k \in \{0, \ldots, 11\}$ and for different choices of shift l and period r, respectively in Cases 1, 2 and 3.

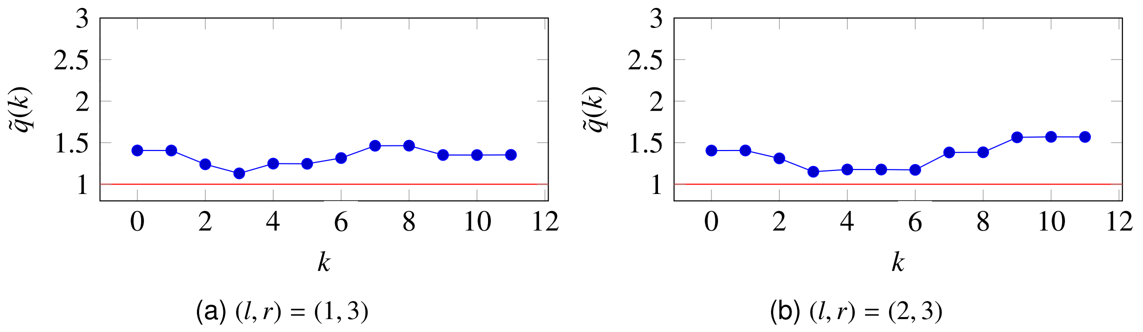
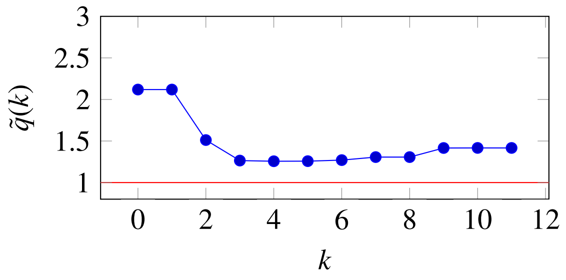


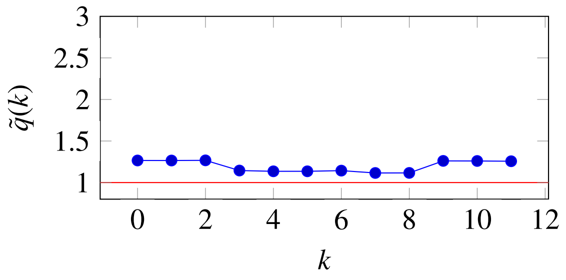
Figure 4.11: Mermin evaluation of states during the QFT of two states $\left|\varphi^{(l,r)}\right\rangle$ from Case 1 of [JH19]

Let us start with general comments.

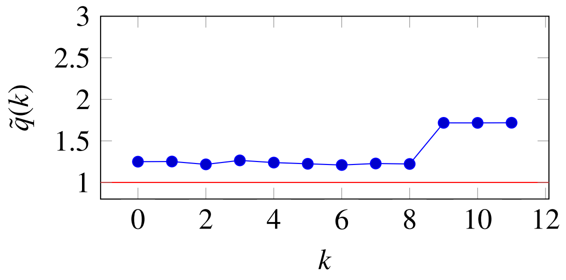
- All examples in Figures 4.11, 4.12 and 4.13 present violations of the Mermin inequality, and the amount of violation evolves during the algorithm. This contrasts with [SSB05] where the authors found almost no evolution of the GME during the QFT. Those statements are not contradictory as entanglement and non-locality are not the same resource but it shows that the Mermin polynomials detect variations of the nature of the states that are not measured by the GME.
- The sets $\{0, 1\}, \{4, 5\}, \{7, 8\}$ and $\{9, 10, 11\}$ for k correspond to states before and after gates of the QFT that do not modify entanglement (Hadamard, SWAP). That explains why the function is constant on those intervals, as it was already the case for the curves $k \mapsto |\Delta_{2222}(\varphi_k^{l,r})|$ in [JH19].
- States corresponding to Cases 1 and 2 of [JH19] violate the classical bound during



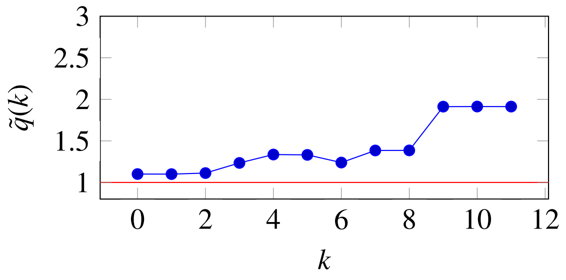
(a) $(l, r) = (0, 3)$



(b) $(l, r) = (5, 3)$

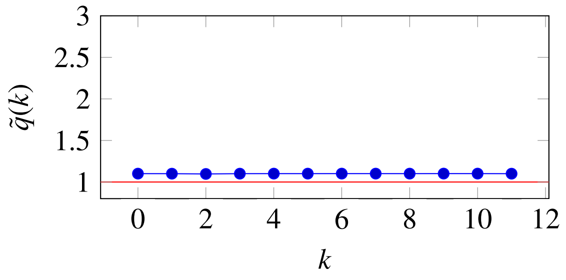


(c) $(l, r) = (11, 1)$

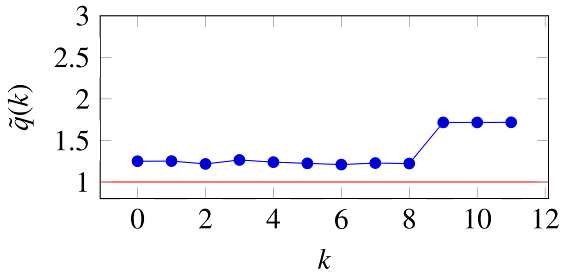


(d) $(l, r) = (9, 1)$

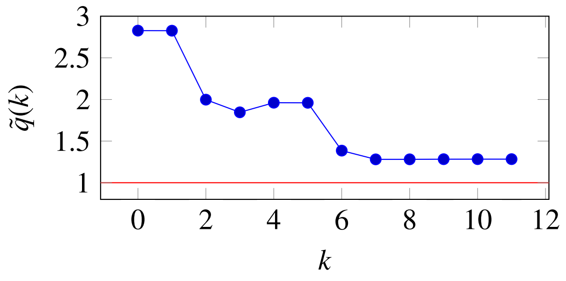
Figure 4.12: Mermin evaluation of states during the QFT of four states $\left|\varphi^{(l,r)}\right\rangle$ from Case 2 of [JH19]



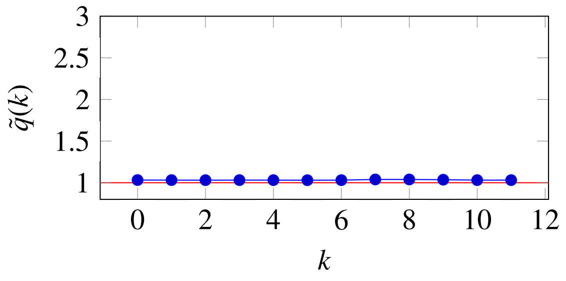
(a) $(l, r) = (2, 2)$



(b) $(l, r) = (0, 11)$



(c) $(l, r) = (0, 15)$



(d) $(l, r) = (1, 1)$

Figure 4.13: Mermin evaluation of states during the QFT of four states $\left|\varphi^{(l,r)}\right\rangle$ from Case 3 of [JH19]

the execution of the QFT. Only some states corresponding to Case 3 produce constant curves with some of them equal to the classical bound (not drawn). It is for instance the case for $(l, r) = (2, 4)$ which is a separable state that remains separable during the algorithm. Figure 4.13 illustrates different possible behaviors of the states in Case 3. These variations were not detected in [JH19] by the evaluation of $|\Delta_{2222}|$.

The amount of violation of non-locality measured during the QFT is not connected to the change of SLOCC classes computed in [JH19] for the same algorithm and input state. Indeed, states in the same SLOCC class reach different values of the maximal violation of the Mermin inequality. For instance, if one considers the periodic states $\left|\varphi^{l,r}\right\rangle$ for $(l,r) = (2,2)$ and $(l,r) = (0,11)$ (Figures 4.13a and 4.13b), it is shown in [JH19] that these two states are SLOCC equivalent (*i.e.* can be inter-converted by a reversible local operation), but their evolution during the QFT is quite different. The value of $\tilde{q}(k)$ fluctuates around 1.10 for $(l,r) = (2,2)$, whereas it is in the interval $[1.65, 2.18]$ for $(l,r) = (0,11)$.

Similarly the cases $(l,r) = (0,15)$ and $(1,1)$ (Figures 4.13c and 4.13d) correspond to two states SLOCC equivalent to $|GHZ_4\rangle$ at the beginning of the algorithm. It is clear for $(l,r) = (0,15)$ because $\left|\varphi^{0,15}\right\rangle = |GHZ_4\rangle$ and $\tilde{q}(k)$ reaches the maximal possible value at the beginning of the algorithm. The maximal violation of Mermin inequality for four qubits is $2\sqrt{2} \approx 2.81$ ($2^{\frac{n-1}{2}}$ for $n = 4$), but this value is nowhere to be approached for $(l,r) = (1,1)$ where the value of $\tilde{q}(k)$ is close to 1 at all steps of the run. In fact the state

$$\left|\varphi^{1,1}\right\rangle = \sqrt{\frac{16}{15}}\,|{+}{+}{+}{+}\rangle - \frac{1}{\sqrt{15}}\,|0000\rangle \tag{4.7}$$

is a state on the secant line joining $|{+}{+}{+}{+}\rangle$ and $|0000\rangle$, as described in Section 4.2.2. This state is indeed SLOCC equivalent to $|GHZ_4\rangle$ but it is closer to a separable state if one considers the GME.

4.4/ Implementation

This section explains the code developed for this chapter and relates it to the notations from Section 4.2. This code can be found at https://quantcert.github.io/Mermin-eval . It uses the open-source mathematics software system SageMath[2] based on Python. The code is a module named `mermin_eval`, and usage examples can be found in the GitHub repository. Note that all the results of this chapter have been double checked, by first being obtained on Maple[3] and then only being generalized on SageMath.

The code is provided and presented for several reasons: so the readers can see how we obtained the results presented in Section 4.3.1.2, and they can reproduce our computations by running the code. But the code can also be extended to other evaluation

[2]http://www.sagemath.org
[3]https://www.maplesoft.com

methods of Grover's algorithm, or adapted to other quantum algorithms, since it is structured in several well-documented functions.

This section is divided in two parts: we first explain the code used for Grover's algorithm in Section 4.4.1, and then the code used for the Quantum Fourier Transform in Section 4.4.2.

4.4.1/ Grover's algorithm implementation

For Grover's algorithm, the main function `grover` is reproduced in Listing 4.1. The parameter `target_state_vector` is the searched state $|\mathbf{x_0}\rangle$. The function first executes an implementation `grover_run` of Grover's algorithm, detailed in Section 4.4.1.1, and stores in the list `end_loop_states` the states after each iteration of the loop $\mathcal{L}$. Then but independently, a call to the function `grover_optimize` (Section 4.4.1.2) optimizes Mermin operator. The result is stored in the matrix `M_opt`. Finally both these results are used to evaluate entanglement after each iteration of $\mathcal{L}$ with a call to the function `grover_evaluate` (Section 4.4.1.3), also responsible of printing the evaluations at each step.

```python
def grover(target_state_vector):
  end_loop_states=grover_run(target_state_vector)

  M_opt=grover_optimize(target_state_vector)

  grover_evaluate(end_loop_states, M_opt)
```

Listing 4.1: Main function for Grover's entanglement study

4.4.1.1/ Execution

The function `grover_run` given in Listing 4.2 takes as input the target state and returns a list of states composed of the states at the end of each loop iteration.

```python
def grover_run(target_state_vector):
  layers, k_opt=grover_layers_kopt(target_state_vector)
  N=len(target_state_vector)
  V0=vector([0, 1] + [0]*(2*N-2))

  states=run(layers, V0)
  end_loop_states=states[0]
  for i in range(k_opt):
    end_loop_states.append(states[2*i+1])

  return end_loop_states
```

Listing 4.2: Function running Grover's algorithm

This function operates in two steps. The first step is to build the circuit for Grover's algorithm, which is achieved by the function `grover_layers_kopt`. The circuit format is described in Sec. 3.1. Then the circuit is executed, which is achieved by the function `run` that returns the list of the states after each layer. The function `run` is also described in Sec. 3.1.

The **`for`**-loop then filters out all the intermediate states which are not at the end of a loop iteration. For example, if we consider Grover's algorithm on three qubits shown in Figure 4.14, we would have the first state $|\varphi_0\rangle$, and the states $|\varphi_3\rangle$ and $|\varphi_5\rangle$ in `end_loop_states`.

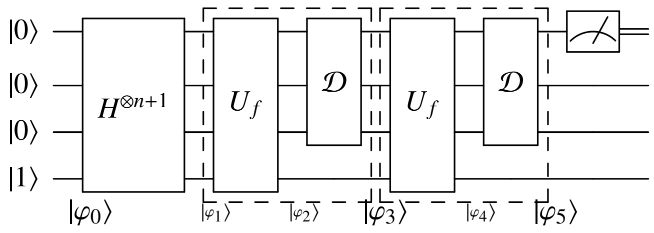


Figure 4.14: End loop counting example

This implementation of the simulation of Grover's algorithm has its limits though. It is computationally expensive to multiply matrices beyond a certain number of qubits. To push it a little further, we used another implementation for Grover's algorithm, less versatile but more efficient. This method is presented in Listing 4.3. In this case, two important differences are first that there is no more use for the ancilla qubit (the last wire in the circuit definition of Grover's algorithm, see Figure 4.1), which divides by two the number of elements in a state vector, and second that almost no matrix multiplication is used. Indeed, the loop is now handled by functions operating directly on the state vector. The first function is `oracle_artificial`, and it only flips the correct coefficient in the running state (this is the behavior explained in Section 4.2.1). The second function `diffusion_artificial` performs the inversion about the mean.

4.4.1.2/ Optimization

The `grover_optimize` function shown in Listing 4.4 computes an approximation of an optimal Mermin operator, as explained in Section 4.3.1.1. The Mermin operator M_n is an implicit function of $(\alpha, \beta, \delta, \alpha', \beta', \delta')$, here implemented as `(a,b,c,m,p,q)`. Because of this, optimizing the Mermin operator is finding the optimal $(\alpha, \beta, \delta, \alpha', \beta', \delta')$ for our Mermin evaluation (which is done using the `optimize` function).

To optimize the Mermin operator, first the state $|\varphi_{ent}\rangle = (|\mathbf{x_0}\rangle + |+\rangle^{\otimes n})/K$ (with K the normalizing factor) is computed and stored in `phi`, then f_{M_n} represented by `M_eval` is used to define $f_{M_n}(|\varphi_{ent}\rangle)$ as `M_phi`. Note that in the mathematical notations, $f_{M_n}(|\varphi_{ent}\rangle)$ is an

```
def grover_run(target_state_vector):
  N=len(target_state_vector)
  n=log(N)/log(2)
  k_opt=round((pi/4)*sqrt(N))
  H=matrix(field, [[1, 1],
                   [1, -1]])/sqrt(2)
  hadamard_layer=kronecker_power(H, n)

  V0=vector([1]+[0]*(N-1))

  V=hadamard_layer * V0
  end_loop_states=[V]

  for k in range(k_opt):
    V=oracle_artificial(target_state_vector, V)
    V=diffusion_artificial(V)
    end_loop_states.append(V)

  return end_loop_states
```

Listing 4.3: Optimized implementation of Grover's algorithm

```
def grover_optimize(target_state):
  n=log(len(target_state))/log(2)
  plus=vector([1,1])/sqrt(2)
  plus_n=kronecker_power(plus, n)
  phi=(target_state + plus_n).normalized()

  def M_phi(a,b,c,m,p,q):
    return M_eval(a,b,c,m,p,q, phi)

  (a,b,c,m,p,q),v=optimize(M_phi, (1,1,1,1,1,1), 5, 10**(-2), 10**2)

  return M_from_coef(n,a,b,c,m,p,q)
```

Listing 4.4: Optimization function for Grover's algorithm

implicit function of $(\alpha,\beta,\delta,\alpha',\beta',\delta')$. This implicit relation is made explicit as `M_phi` is a function of `(a,b,c,m,p,q)`.

The `optimize` function takes as input a function (here `M_phi`), a first point to start the optimization from (here `(1,1,1,1,1,1)`), the step sizes bounds and a maximal number of iterations on a single step (here 10^2). The random walk starts with a step size of 5 and ends with a step size of 10^{-2}.

The optimization function proceeds with a random walk. It iterates until it finds a local maximum (for all points p in a neighborhood around the point found p_{opt}, their evaluation by the function given as the first parameter is less than the evaluation of the point found $f(p) \leq f(p_{opt})$). To find this optimum, the process starts from an arbitrary point (given as an argument) and at each step, an exploration of the space is done around the current point until the evaluation on the argument function increases. If an increase cannot be found before the fixed maximal number of iterations, the step size is reduced, otherwise the same step is repeated with the same step size. The function ends when the step size reaches the fixed minimal size of the steps.

Remark 5: *This optimization can be expensive, so to speed up the calculation, a memoization step is hidden here: if* `(a,b,c,m,p,q)` *has already been computed for* `target_state`*, this result has been stored on disk at this point and is now loaded.*

4.4.1.3/ Evaluation

The function `grover_evaluate` shown in the Listing 4.5 is the simplest of the three: it computes $f_{M_n}(|\varphi_k\rangle) = \langle\varphi_k|M_n|\varphi_k\rangle$ for each $|\varphi_k\rangle$ in the `end_loop_states` list with M_n here being `M_opt`, and prints them.

```
def grover_evaluate(end_loop_states, M_opt):
  for state in end_loop_states:
    print((state.transpose().conjugate()*M_opt*state))
```

Listing 4.5: Evaluation function for Grover's algorithm

To overview the code as a whole, we can exhibit the link with Figure 4.5. For this figure, each graph has been obtained by using a code line such as in Listing 4.6 (where `string_to_ket` is a function used to convert a string of a specific format into a vector, in this case the vector $|0000\rangle$). So, for four qubits, we set the target state as $|0000\rangle$, for five qubits as $|00000\rangle$, and so on. This is enough for symmetry reasons (searching for $|1001\rangle$ instead of $|0000\rangle$ yields similar results).

```
>>> grover(string_to_ket("0000"))
0.173154027401573
1.01189404012534
-0.469906068136016
```

Listing 4.6: Mermin evaluation in Grover's algorithm example

4.4.2/ Quantum Fourier Transform implementation

For the QFT, the main function `qft` is reproduced in Listing 4.7. The parameter `state` is the state ran through the QFT, generally a periodic state $\left|\varphi^{l,r}\right\rangle$ generated by the function `periodic_state` (Listing 4.8). The function `qft` first calls an implementation `qft_run` of the QFT, detailed in Section 4.4.2.1, and stores the computed states in the list `states`. Then the states are directly evaluated. The important difference compared to Grover's algorithm implementation is the fact that we are not using a separate optimization step, the optimization process is included in the evaluation process: each evaluation requires an optimization. The evaluation process is thus performed by the function `qft_evaluate` (Section 4.4.2.2), printing the evaluation as well.

```
def qft_main(state):
  states=qft_run(state)
  return qft_evaluate(states)
```

Listing 4.7: Main function for QFT entanglement study

```
def periodic_state(l,r,nWires):
  N=2**nWires
  result=vector(N)
  for i in range(ceil((N-l)/r)):
    result[l+i*r]=1
  return result.normalized()
```

Listing 4.8: Function used to generate the periodic state $\left|\varphi^{l,r}\right\rangle$

4.4.2.1/ Execution

The function `qft_run` (Listing 4.9) uses the same circuit format as `grover_run` presented in Section 4.4.1.1. This circuit is built by `qft_layers` (Listing 4.10) and run by `run`. In this case however, the states do not need to be filtered, resulting in an almost trivial `qft_run` function.

```
def qft_run(state):
  layers=qft_layers(state)
  states, _=run(layers, state)
  return states
```

Listing 4.9: Function running the QFT

The `qft_layers` function uses two functions not detailed here. `swap` returns a matrix corresponding to the swap of two wires `wire1` and `wire2` and the identity on the other wires concerned. The `R` method returns the controlled rotation of angle $e^{\frac{2i\pi}{2^k}}$, with the rotation being performed on the wire `target` controlled by the wire `control`. The two matrices built by these functions have a size of `2**size`. With these two functions, `qft_layers` builds the circuit for the QFT using `R` on the whole width of the circuit when a rotation is needed and using `swap` only at the end to build the global swap (in fact, `swap` is also used in `R` and that is the reason why this implementation of swap on two wires has been chosen instead of a more general arbitrary permutation gate).

```
def qft_layers(state):
  def swap(wire1,wire2,size):
    ...
  def R(k,target,control,size):
    ...
  H=matrix(field, [[1,  1],
                   [1, -1]])/sqrt(2)
  I2=matrix.identity(field, 2)
  nWires=log(len(state))/log(2)
  layers=[]

  for wire in range(nWires):
    layers.append([I2]*wire + [H] + [I2]*(nWires-wire-1))
    for k in range(2, nWires-(wire-1)):
      layers.append([R(k, wire, k+(wire-1), nWires)])

  global_swap=matrix.identity(field, 2**nWires)
  for wire in range(nWires/2):
    global_swap *=swap(wire, nWires-1-wire, nWires)
  layers.append([global_swap])

  return layers
```

Listing 4.10: Function building the circuit of the QFT

4.4.2.2/ Evaluation

In this case again, the evaluation is conceptually simpler than in Grover's algorithm. Indeed, since the optimization needs to be performed for each evaluation, the result printed at each step is simply the optimal point reached by the `optimize` function (the same as described in Section 4.4.1.2). In this case, a notable difference in the usage of `optimize` is the presence of `3*n*2` coefficients. This is explained by the fact that, this time, we do not want a trend for the evaluation's evolution and a "good enough" M_n. This means that we do not stand satisfied by the constant $a_n = \alpha X + \beta Y + \delta Z$ but we have α, β and δ variable as explained in 4.3.2.1 (where they become $(\alpha_i)_{1 \le i \le 6n}$).

Because of this, the function `M_func` (Listing 4.11) we optimize is now calling `M_eval_all` instead of `M_eval`. The difference is that `M_eval` took only 3×2 coefficients to compute M_n with fixed $a_i = \alpha X + \beta Y + \delta Z$ and $a'_i = \alpha' X + \beta' Y + \delta' Z$, whereas this time the coefficients of a_i and a'_i are variable, thus `M_eval_all` takes as arguments two lists of triples `_a_coefs` and `_a_prime_coefs` (each triple encoding one a_i or a'_i). The function `coefficients_packing` reshapes as two lists of triples the flat list of reals that `M_func` requires as input.

```
def qft_evaluate(states):
  n=log(len(states[0]))/log(2)
  for state in states:
    rho=matrix(state).transpose()*matrix(state)

    def M_func(_a_a_prime_coefs):
      _a_coefs, _a_prime_coefs=coefficients_packing(_a_a_prime_coefs)
      return M_eval_all(n, _a_coefs, _a_prime_coefs, rho)

    _,value=optimize(M_func, [1]*3*n*2, 5, 10**(-2), 10**2)

    print value
```

Listing 4.11: Evaluation function for the QFT

4.4.3/ Implementation recap

Finally, to conclude this section, we recall the functions reusable in a general context, the `run` function can be used for general purpose quantum circuit simulation and the Mermin evaluation process can be used for arbitrary state entanglement evaluation. An issue previously mentioned was the correctness between the process and the simulation, and here this issue is tackled by structured and clear code. This structure also helps the code to be more modular, for instance, if the user wants to change the optimization method for more speed or precision, it can be easily achieved.

Remark 6: *Note that the actual implemented functions have additional parameters that are ignored here for simplicity's sake. For example, each function has a verbose mode, to display more information about its run.*

4.5/ Conclusion

In this chapter, we have shown that both Grover's algorithm and the QFT generate states that violate Mermin's inequalities. We provided, for different settings, curves measuring the evolution of the non-local behavior of the states through the algorithms. Evaluation of Mermin polynomials detects entanglement when it violates the classical bound and we compared our numerical results on non-locality evolution with the evolution of values obtained from several measures of entanglement for the same algorithms. Understanding the connection between entanglement and non-local properties of quantum states is a difficult question and we did not intend to provide new theoretical perspectives on this subject. Instead our goal was more to focus on an operational level by studying how specific properties of quantum states generated by those algorithms behave.

This work is a step towards contributions in quantum program verification, by checking state properties, such as entanglement or violation of classical Bell inequality, or temporal properties, such as the increase or decrease of a quantity related to non-locality, during the execution of a quantum program. In the present work we check properties during the execution of the program, for a fixed number of qubits. A promising possibility is to check properties statically, without executing the program and once for all numbers of qubits. A theoretical foundation for this static verification is the quantum Hoare logic [Yin11], an adaptation of the Hoare logic [Hoa69] to quantum programs. Mermin polynomials studied in this chapter seem promising to check properties during program execution, since Mermin evaluation corresponds to an experimental measurement that could be performed on a quantum computer (see for instance [AL16] for examples of Mermin evaluation on a 5-qubit computer, Chap. 5 presents another such implementation).

5

Implementation aimed at execution on a quantum processor

One of the motivations for developing Mermin's polynomials as a tool for studying entanglement is that they have the potential to be executed on an actual quantum computer. This means that the simulator developed to obtain the results presented in Chap. 3 and 4 can now be replaced by a counterpart running on a quantum computer. In order to do this, we turn to the IBM quantum experience, and more specifically to the Qiskit library, an extensive python library offering an interface with either a local simulator, a cloud accessible simulator, or the quantum processors made accessible through internet by IBM. This work was done in cooperation with Grâce Amouzou, another PhD student working with Frédéric Holweck. This work was in fact not published on my side, but Grâce published some results coming from our common work [ABJ+20].

In this chapter, we first present the Qiskit ecosystem in Sec. 5.1. We then describe how we compute the Mermin evaluation on Qiskit in Sec. 5.2, and we finally show how we use it to reproduce our study of entanglement for the QFT in Sec. 5.3.

5.1/ The Qiskit ecosystem

The first guided approach to IBM quantum experience is via the composer (available at https://quantum-computing.ibm.com/composer). It is a visual interface to create quantum circuits such as the one depicted in Fig. 5.1.

But as mentioned before, the circuitry has important practical limitations. The first tool to circumvent this limitation is the QASM language, also available on the IBM quantum experience. But since this language aims at mimicking quantum circuitry, most of the problems of the latter remain. The next step suggested by IBM's tutorial is to use Qiskit,

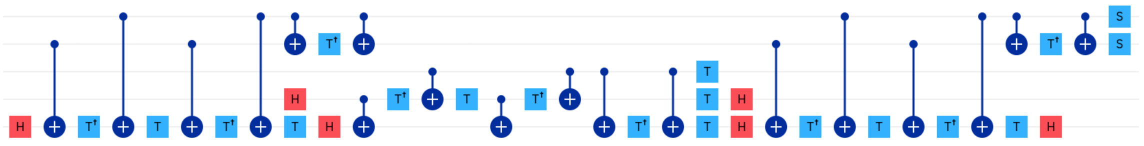
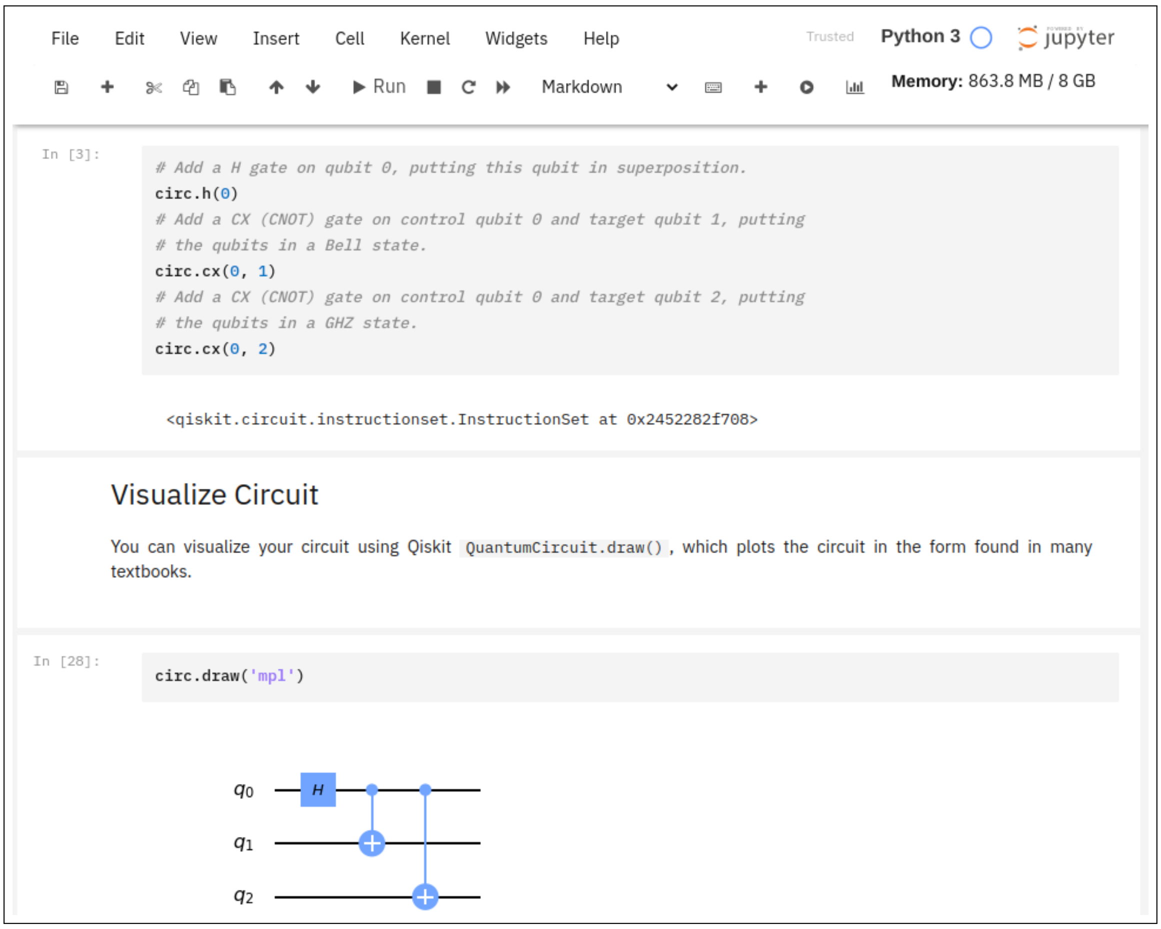


Figure 5.1: Decomposition of the Toffoli gate using IBM's composer

with Jupyter suggested as the method of choice to run the Qiskit code, as shown in Fig. 5.2. But since Qiskit is a Python library, the user has all the power of the python language to play with. Given this, I implemented the code to run the QFT on IBM's quantum computer, and analyze it using Mermin's polynomials. The corresponding code is available at https://quantcert.github.io/Mermin-hypergraph-states/Qiskit/.



```
# Add a H gate on qubit 0, putting this qubit in superposition.
circ.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1, putting
# the qubits in a Bell state.
circ.cx(0, 1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 2, putting
# the qubits in a GHZ state.
circ.cx(0, 2)
```

```
<qiskit.circuit.instructionset.InstructionSet at 0x2452282f708>
```

Visualize Circuit

You can visualize your circuit using Qiskit `QuantumCircuit.draw()`, which plots the circuit in the form found in many textbooks.

```
circ.draw('mpl')
```

Figure 5.2: Introduction to Qiskit on Jupyter

5.2/ Mermin implementation on Qiskit

I started by implementing a wrapper for the circuit execution in order to simplify this process. A simplified version of this wrapper is shown in listing 5.1 where `Aer` and `IBMQ` are classes from the Qiskit library. `Aer` provides the simulator and `IBMQ` the access to IBM's quantum experience computation nodes (classical simulators and quantum processors).

```
def runCircuit(qc, simulation=True, local=True, shots=1024):
  n=qc.num_qubits
  if local:
    backend=Aer.get_backend('qasm_simulator')
  else:
    backend=least_busy(IBMQ.get_provider(group='open').backends(
      simulator=simulation,
      filters=lambda x: x.configuration().n_qubits > 4))
  job_exp=execute(qc, backend=backend, shots=shots)
  return job_exp.result()
```

Listing 5.1: Wrapper around circuit execution

For a given circuit, we compute the Mermin evaluation $\langle\varphi_k|Mn|\varphi_k\rangle$ of each of its intermediate steps using the `evaluate_polynomial` function presented in listing 5.2. This function takes as inputs the number of qubits, the circuit to be analyzed (each intermediate step will be evaluated), and the coefficients used to build the families of observables (a_i) and (a'_i) involved in the implementation of the Mermin polynomial, as presented in Sec. 2.4.1.

```
def evaluate_polynomial(n, circuit, a_a_p_coeffs,
        shots=1024, is_simulation=True, local=True):
    mermin_polynomial=mermin_IBM(n)
    total_result=0
    for i in range(2 ** n):
        if mermin_polynomial[i] !=0:
            measure_monomial=evaluate_monomial(n,i,circuit,a_a_p_coeffs,
              shots=shots,is_simulation=is_simulation,local=local)
            total_result+=mermin_polynomial[i]*measure_monomial
    return abs(total_result)
```

Listing 5.2: Function computing the Mermin evaluation of a state prepared by a circuit.

This function itself uses the function `evaluate_monomial`. This is due to the fact that $\langle\varphi|M_n|\varphi\rangle$ cannot be evaluated directly. In order to evaluate it, one has to evaluate each monomial composing the polynomial M_n.

But before the explanation of the monomials computation, one should understand how the Mermin polynomial itself is computed, because this is a quite interesting trick. First, as a reminder, $M_n = \sum_{i=0}^{2^n-1} \alpha_i a_1^{\beta_{1,i}} \otimes \ldots \otimes a_n^{\beta_{n,i}}$ with $\alpha_i \in \{0\} \cup \{\frac{1}{2^k}, k \in [0..n]\}$ and β is either the prime mark " $'$ " or nothing. Furthermore, there is no need to know the a_i's and a'_i's for computing the Mermin polynomial at first. Indeed, we first compute the coefficients α_i using the formula defining the Mermin polynomial, given in Eq. 2.1. This is done in the function `mermin_IBM`, with the i^{th} entry of the resulting array corresponding to the monomial $a_1^{\beta_{1,i}} \otimes \ldots \otimes a_n^{\beta_{n,i}}$ where $\beta_{k,i}$ is " $'$ " if the k^{th} digit of i is 1 and " " otherwise. Another trick used in `mermin_IBM` but not explicitly shown here is that the coefficients of $M_n(a', a)$ are simply the coefficients of $M_n(a, a')$ in reverse order. This allows us to have a single recursive function to compute these coefficients, instead of a double recursion.

Now that the coefficients of the monomials are computed, let us recall that the rest of the information is contained in the coefficients defining the a_i's and a'_i's. Indeed, as explained in Chap. 4, each a_j is defined as $a_j = \alpha_j X + \beta_j Y + \gamma_j Z$. Those coefficients are stored in the variable `a_a_p_coeffs` as a pair of lists of triples. The first element of the pair contains the coefficients for the a_i's and the second one the coefficients for the a'_i's. Each of these elements is a list of triples, where each element of the list contains the three coefficients defining the corresponding a_i or a'_i .

```
def evaluate_monomial(n, n_measure, circuit, a_a_p_coeffs, shots,
        is_simulation=True, local=True):
    circuit_size=circuit.num_qubits
    circuit_aux=QuantumCircuit(circuit_size,n)
    basis_change.U3_gates_placement(n,n_measure,a_a_p_coeffs,circuit_aux)
    circuit_mesure=circuit + circuit_aux
    result=run.runCircuit(
        circuit_mesure,shots=shots,simulation=is_simulation,local=local)
    measure_monomial=measures_exploitation(result,shots)
    return measure_monomial
```

Listing 5.3: Function returning the average value of a state prepared by a circuit measured by a given monomial.

We use this information to compute the monomials as shown in listing 5.3. This function prepares the total circuit, made of both the circuit preparing the evaluated state and the evaluation circuit. The evaluation circuit is `circuit_aux` in the body of the function, and is prepared using the function `U3_gates_placement`. Indeed, the a_n and a'_n observables can be created using the gate $U_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix}$. This is because, if we set $\theta = \arccos(\gamma)$ and $\phi = \arccos(\alpha/\sin(\theta))$ then $\alpha X + \beta Y + \gamma Z = U_3(\theta, \phi, -\phi - \pi)$. Note that in this formula, β does not appear: this is in fact not so strange since $|\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1$. To go into more details, the formula must be adjusted on edge cases, the total computation is shown in listing 5.4.

```
def mermin_coeffs_to_U3_coeffs(x, y, z):
    theta=np.arccos(z)
    phi=0 if np.sin(theta)==0 else np.arccos(x/np.sin(theta))
    if y/np.sin(theta) < 0:
        phi=- phi
    return theta, phi
```

Listing 5.4: Function converting the coefficients of a_k to angles to feed to the function building the corresponding U_3 gate.

The measure is performed in the basis associated to the Z Pauli matrix, so an odd number of 1s in the result would yield a global measurement of -1 and an even number of 1s would yield a global measurement of 1. To get the average of the results, one then has to count positively each result that yielded an even number of 1s and negatively each result that yielded an odd number of 1s. This operation is performed by the `measure_exploitation` function given in listing 5.5.

```python
def measures_exploitation(measures_dictionary, shots):
    results=0
    for measure in measures_dictionary:
        sign=1 if measure.count('1') % 2==0 else -1
        results +=sign*measures_dictionary[measure]
    return (even_results - odd_results) / shots
```

Listing 5.5: Function computing the results interpretation for a monomial.

5.3/ QFT on Qiskit

With this, we have finished our diving into the Mermin evaluation in Qiskit, but we haven't seen a concrete example yet. We have created several examples, available in the `example` section of the git repository, but I will only present a single one here: the Mermin evaluation along the QFT. In order to do this, we need to have a circuit building the state at each step of the QFT, *i.e.* a truncated QFT circuit. This is precisely the role of the function `build_QFT_0_to_k`, presented in listing 5.6. This function simply adds successively each gate of the QFT, and breaks when the number of gates is equal to the desired number of gates.

One can easily see the result of the function using the `draw` method of the Qiskit circuits as such: `build_QFT_0_to_k(4,3).draw()` which prints the figure shown in Fig. 5.3.

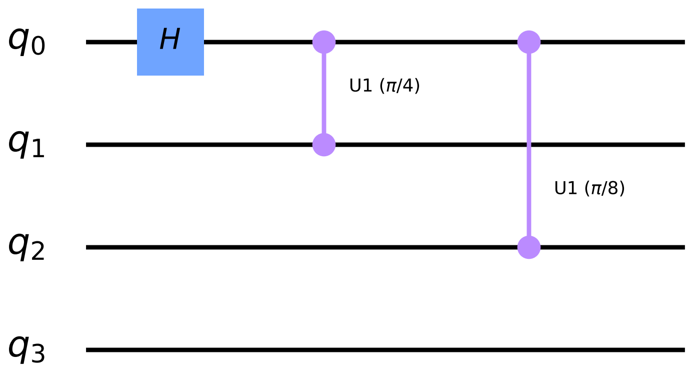


Figure 5.3: Partial QFT containing the first 3 gates of the QFT on 4 wires

At this point, we load the optimized coefficients, generated by the code presented in Chap. 4, and run the entanglement analysis of the QFT on IBM's quantum processor by

```python
def build_QFT_0_to_k(nWires, k):
    partial_QFT=QuantumCircuit(nWires)
    if partial_QFT.size==k:
        return partial_QFT
    for wire in range(nWires):
        partial_QFT.h(wire)
        if partial_QFT.size==k:
            return partial_QFT
        for inlayer_nb in range(2, nWires-(wire-1)):
            theta=np.pi/(2**inlayer_nb)
            ctl=wire
            target=inlayer_nb+(wire-1)
            partial_QFT.cu1(theta, ctl, target)
            if partial_QFT.size==k:
                return partial_QFT
        partial_QFT.barrier()
    init_order=nWires
    final_order=[nWires-1-wire for wire in range(nWires)]
    SWAP=Permutation(init_order, final_order)
    result=partial_QFT + SWAP
    return result
```

Listing 5.6: Function creating the truncated QFT circuit containing the first k gates of the circuit.

calling the main QFT function shown in listing 5.7.

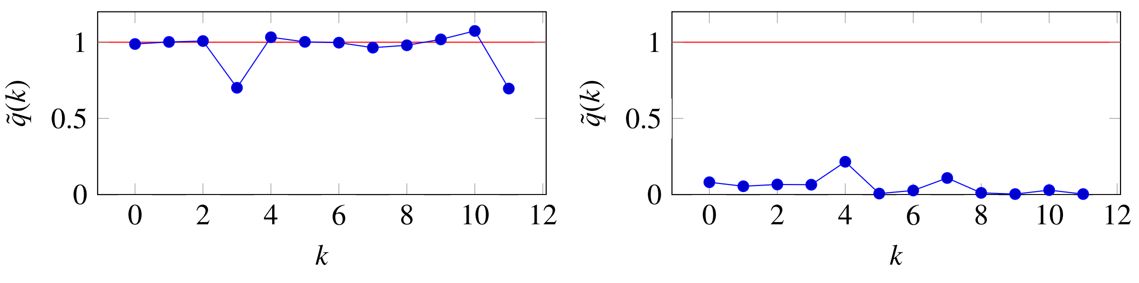
```python
def main(l,r,nWires,optimization_filepath,local=True,simulation=True):
    s=periodic_state(l,r,nWires)
    preparation_circuit=QuantumCircuit(nWires).initialize(s,range(nWires))
    for k in range(QFT_length(nWires)+1):
        coeffs=get_coef_from_optimization_file(optimization_filepath,k)
        QFT_partial=build_QFT_0_to_k(nWires,k)
        value=evaluate_polynomial(nWires,preparation_circuit+QFT_partial,
            coeffs,is_simulation=simulation,local=local and simulation)
        print(value)
```

Listing 5.7: Main function computing the Mermin evaluation along each step of the QFT.

Once again, the process presented here is slightly simplified, the complete example is available on https://quantcert.github.io/Mermin-eval with code documentation, installation instructions, and many examples.

The only step left is to present the result. The Mermin measurements are represented in Fig. 5.4 showing both the simulation on Qiskit and the run on IBM's quantum processors.

This is the disappointing part: to this day, the call of `IBMQ`, the Qiskit module in charge of interacting with IBM's cloud infrastructure, is quite unstable, and network errors are not rare. This, added to the noise in quantum computation, yield results that are very



(a) simulated (b) quantum

Figure 5.4: Simulated and actual quantum runs of our implementation of QFT of $\left|\varphi^{1,1}\right\rangle$ on Qiskit

far from the theoretical ones. A reassuring point though is that the algorithm performs flawlessly using IBM's simulator and the function presented in this chapter instead of our simulator presented in Chap. 4. This leads to believe that both implementations achieve the desired behavior, and that the strange experimental results are only due to quantum noise and network errors. This is a good example of how such measures could be useful to detect the proper implementation of algorithm, or the good behavior of the machine on which the algorithm runs. At this moment though, with our experimental setup, it is more expensive to perform the Mermin measure that to run the original algorithm, so this may be a tool for later use, but for now, it is only a proof of concept.

III

Quantum geometries

This introduction serves to introduce the works presented in this Part. The emphasized words in this introduction are fully defined in the three chapters of this Part.

The first quantum property we studied, entanglement, showed us possible characterizations of the studied quantum algorithms. This property often presented to new comers as a defining property of quantum computing is commonly put in evidence using Bell non-locality, but those experiences can all be mapped to experiences putting contextuality in evidence [Cab21]. Knowing this, the study of this new property has a crucial role if we intend to reason on quantum properties. We approach this property from the point of view of quantum geometries, a special kind of finite geometries linked to quantum contextuality.

Finite geometries are also mentioned in the literature as block designs. I use this term in Chap. 6 because this chapter follows previous articles using this terminology, but in the following chapters, I simply call them geometries.

Chap. 6 presents a method found in the literature to build block designs/finite geometries, as well as a thorough check of the validity of this method. Chap. 7 and 8 push further the exploration of finite geometries construction, but this time based on binary symplectic polar spaces. This work is much more implicated in the study of contextuality, since the built geometries are immediate candidates for contextual geometries. In Chap. 7, I explain how I generated contextuality proofs in Magma using the symplectic space, and in Chap. 8 I present a joint work with Metod Saniga, in which we classify certain subgeometries of the symplectic spaces, when the points of these spaces canonically encode n-qubit observables, for $n = 2, 3, 4$.

6

Formalization and validation of a block design construction method

This chapter is the translation of an article written in French for a French conference [Cd19]. It has been initially written with Jessy Colonval, a master student at this time, currently working as a Ph.D. student. We thank Michel Planat for his help on the matter.

6.1/ Introduction

In the domain of experimental mathematics, the researchers often write code to discover new results. But this is one of the domains where reproducibility could greatly be improved, since, most often, the programs are either not publicly available or are not sufficiently documented or structured to allow the reader to run said program and reproduce the results. Reproducibility is a major stake that must also be considered in computational mathematics [SBB+12], in particular to ease the verification and the extension of published results.

A mathematical theorem is often a general statement that must be proven, since the calculation allows to validate only a limited fragment of an infinity of cases covered by this statement. This theorem can be verified by a third party, in a proof assistant, if its publication is accompanied by a formal demonstration. However, the construction of a formal proof is a difficult task, since it requires a formalization of all the mathematical concepts underlying the theorem, and the mastery of a proof assistant. The formal proof of the *odd order theorem* [GAA+13] is a good example: it mobilized many researchers over a long period of time. Comparatively, the test of a large number of instances of a theorem is often much simpler to implement and can bring a sufficient level of confidence in the correction of this theorem.

We propose to apply intensive testing to computational mathematics, as well as other good programming practices, such as structuring, code factoring, documentation and free distribution of the source code. In particular, we aim at theorems dealing with infinite families of mathematical structures with a size. In this case, we propose to verify the theorem for all structures, by increasing size, up to a maximal size considered as sufficient to give confidence in the correctness of the theorem.

This work is part of the research on finite geometries called *quantum* geometries, because of their link to quantum contextuality. Planat et al. [PGHS15] have shown how to construct these geometries from groups of permutations, but without publishing a program for this construction. These geometries are special cases of *block designs* (defined in Sec. 6.2) and the article of Planat et al. presents a method to build them.

A bibliographical search on the origin of this method led us to a previous reference [KM02], which presents a simpler method, which builds block designs from primitive permutation groups. This article is completed by a program, but the latter does not contain any code to validate this method. We formalize this method, and then validate it by enumeration. More precisely, we validate that the block designs built according to this method have all the characteristics announced in a proposition of this article. We use the Magma [BCP97] environment, composed of a structured imperative language and a large library of mathematical functions, in particular for group theory and *designs*. Our source code is freely distributed on GitHub[1].

Sec. 6.2 presents our implementation of the block design construction method of Key and Moori [KM02]. Sec. 6.3 deals with our intensive testing of this implementation.

6.2/ Block designs construction

Through the following proposition, Key and Moori [KM02] define a method for constructing block designs:

> Excerpt from Prop. 1 on page 3 of [KM02]
>
> **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 $\mathcal{B}$ forms a self-dual 1-$(n, |\Delta|, |\Delta|)$ design with n blocks [...].*

Sec. 6.2.1 explains this proposition, but the reader unfamiliar with these concepts can admit its content. Sec. 6.2.2 presents our implementation of the computational content of this proposition.

[1] https://quantcert.github.io/Designs

6.2.1/ Definitions

Prop. 2 holds for any natural number n and for any group of permutations G on a finite set Ω of cardinality n, noted here as the set $\{1,\dots,n\}$. By nature, this group G is finite. The application of the permutation g of G to the element α of the set Ω is denoted $g \bullet \alpha$. The image of a part Δ of Ω by the permutation g of G is noted $\Delta^g =_{\text{def}} \{g \bullet x : x \in \Delta\}$. The following definitions from group theory and block designs theory allow us to understand the rest of the proposition, where $|\Delta|$ denotes the cardinality of the orbit Δ.

The group of permutations G on Ω is transitive if, for all elements x and y of Ω, there exists a permutation g of G such that $g \bullet x = y$. The group G is primitive if it is transitive and if it preserves no non-trivial partition of Ω (the trivial partitions of Ω are the partition $\{\Omega\}$, whose only element is Ω, and the partition $\{\{x\} : x \in \Omega\}$, composed of all singletons in Ω). The stabilizer $G_\alpha =_{\text{def}} \{g \in G : g \bullet \alpha = \alpha\}$ of an element α of Ω (*under the action of* G) is the set of permutations of G which leave α invariant under their action. The orbit $O_x =_{\text{def}} \{g \bullet x : g \in H\}$ of an element x of Ω according to a group H of permutations on Ω is the set of the images of x by the permutations of H.

A block is a subset of Ω and a block design is a set of blocks. An incidence structure is a triple $\mathcal{D} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ where $\mathcal{P} = \{1,\dots,n\}$ is a set of elements, $\mathcal{B} = \{1,\dots,b\}$ indexes the blocks in a block design on $\mathcal{P}$ and $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{B}$ is an incidence relation, which defines the membership of an element to a block. Its *dual* structure is the incidence structure $\mathcal{D}^t =_{\text{def}} (\mathcal{B}, \mathcal{P}, \mathcal{I}^t)$ where $(y,x) \in \mathcal{I}^t$ if and only if $(x,y) \in \mathcal{I}$. The incidence relation allows to associate a block design with an incidence structure, and to define the dual of a block design. A block design is symmetric if it has as many elements as blocks. It is self-dual if it is moreover isomorphic to its dual. A t-(v,k,λ) block design has v blocks of cardinality k, such that each subset of λ blocks has exactly t distinct elements in common.

Prop. 2 states that, for any element α of Ω and for any orbit Δ of the stabilizer of α under the action of G different from the orbit $\{\alpha\}$, the block design $\mathcal{B} =_{\text{def}} \{\Delta^g : g \in G\}$ contains n blocks and has the regularity of a 1-$(n, |\Delta|, |\Delta|)$ self-dual system. The following example illustrates these definitions from a given group.

Example 3: *Let* $n = 5$ *and* $G = \{Id, (1,2,3,4,5), (1,5,4,3,2), (1,3,5,2,4), (1,4,2,5,3), (2,5)(3,4), (1,2)(3,5), (1,5)(2,4), (1,3)(4,5), (1,4)(2,3)\}$ *a primitive permutation group on* $\Omega =_{def} \{1,2,3,4,5\}$. *In addition to the identity permutation, noted* Id*, its permutations are given as their product of disjoint cycles. The fixed points (the cycles of length 1) are not explicitly written. With* $\alpha = 1$*, the stabilizer of* G *on* α *is the group* $G_1 = \{Id, (2,5)(3,4)\}$*. The orbits of* G_1 *are the sets* $\{1\}$*,* $\{2,5\}$ *and* $\{3,4\}$*. For* $\Delta = \{2,5\}$*, the block design is* $\mathcal{B} = \{\{2,5\},\{1,3\},\{1,4\},\{2,4\},\{3,5\}\}$*. It is a* 1-$(5,2,2)$ *block design on* 5 *elements, composed of* 5 *blocks of cardinality* $k = 2$*, where each element* ($t = 1$) *is present in exactly* $\lambda = 2$ *distinct blocks.*

6.2.2/ Implementation

Magma has functions to build primitive groups [BCP97, `PrimitiveGroups`], orbits of a group [BCP97, `Orbits`], the stabilizer of an element by the action of a group [BCP97, `Stabilizer`] and the creation of a block design from an incidence structure [BCP97, `Design`], but does not propose an implementation of the method of Key and Moori to build block designs from a primitive permutation group.

The article [KM02] is completed by Magma code (in its appendix) which does not formalize the computational content of Prop. 2 by a function, but only applies it to two particular cases of primitive groups of permutations. Thus, this code is not structured. Moreover, it does not contain any instruction to validate the property of the constructed systems announced in Prop. 2. Finally, the results returned by this code are mixed with the latter, without being placed in a comment block. All these points make the code quite unconvincing and not very reusable.

This is why we propose an implementation of Prop. 2 by two functions, reproduced in the listings 6.1 and 6.2. In order to facilitate their use, they are commented following the Javadoc formatting (since Magma does not offer a tool allowing to generate a documentation[2]). Magma variables are named and the functions are structured in such a way that the code resembles as much as possible to the text of Prop. 2.

The listing 6.1 presents a function that computes the set of orbits Δ defined in Prop. 2. It takes as parameter the primitive permutation group G and returns an associative array `Deltas` of the orbits Δ indexed by the corresponding α. For all elements α of the set $\Omega = \{1, \ldots, n\}$, the loop computes the stabilizer group G_{alpha} in the variable `Galpha`, and its orbits in the variable `orbits`, then associates to index α the sequence of orbits Δ different from the singleton $\{\alpha\}$. The orbits Δ are converted into sets thanks to the Magma function `IndexedSetToSet` [BCP97] so that the results are in usable form for the creation of some Magma objects useful for the upcoming functions.

Listing 6.1: Function computing the orbits Δ from a primitive permutation group.

```
/**
 * 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);
```

[2]There is however a GitHub project, "magdoc", aiming at fulfilling this functionality

```
    orbits := Orbits(Galpha);
    Deltas[alpha] := { IndexedSetToSet(Delta) : Delta in orbits | Delta ne { alpha } };
  end for;
  return Deltas;
end function;
```

The listing 6.2 presents a function that builds block designs from a primitive group. It takes as parameter a primitive permutation group G and returns an associative array of block designs indexed by the associated orbit Δ. The orbits Δ are constructed using the previous function, then each block design is constructed by applying each permutation of the group G to an orbit Δ. The result is associated to the index Δ in the associative array `blocks`.

Listing 6.2: Function for computing block designs from a primitive group of permutations.

```
/**
 * 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 orbits delta
 *    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;
```

6.3/ Validation

A bibliographic search revealed the existence of a correction of Prop. 2, by the same authors [KM08]. This correction does not question the computation of block designs but only their nature: they must be symmetric, and not self-dual, as summarized in Prop. 3.

> Extract from Prop. 1, page 1 of [KM08]
>
> **Proposition 3:** *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. [...]*

This section tries to validate Prop. 2 and 3 using an exhaustive test, bounded by the number of primitive groups used. We first present three Boolean functions that implement the properties of the block designs of Prop. 2 and 3.

The listing 6.3 presents a Boolean Magma function that characterizes the t-(v, k, λ) block designs. It converts the set of sets `blocks` into an incidence structure, to determine with the Magma function `IsDesign` [BCP97] if it is a t-block design. If so, it builds this block design to extract its characteristics with the Magma function `Parameters` [BCP97], then it checks that these characteristics are indeed those expected.

Listing 6.3: Characteristic function of a t-(v, k, λ) block design.

```
/**
 * Characterization of t-(v,k,lambda) block designs.
 *
 * @param blocks::Set The design blocks.
 * @param t::RngIntElt The number of distinct elements 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 the t elements.
 * @return BoolElt Indicates that blocks 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;
```

Prop. 2 indicates that the constructed block design is a self-dual 1-$(n, |\Delta|, |\Delta|)$ block design. The listing 6.4 presents a boolean Magma function to check this condition. The function has as parameters the number of elements of the block design, the block design itself and its orbit Δ. It returns `true` if the constructed block design is a 1-$(n, |\Delta|, |\Delta|)$ block design (checked with the previous function) and if it is self-dual (checked with the function Magma `IsSelfDual` [BCP97]).

Listing 6.4: Properties of the block designs of Prop. 2.

```
/**
 * 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 if 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;
```

Magma already has a function `IsSymmetric` [BCP97] which characterizes a symmetric block design, but according to a different definition than the one used by Key and

Moori [KM02]. The Magma documentation does not specify it, but after some tests, it seems that Magma implements the BIBD definition of symmetry. A BIBD (Balanced Incomplete Block Design) is a 2-(v, k, λ) block design. A BIBD is symmetric if it has as many elements as blocks [Col10]. Thus, Magma considers as symmetrical only block designs whose parameter t is equal to 2.

The listing 6.5 presents two Boolean Magma functions to verify the condition of Prop. 3 according to these two interpretations of the symmetry definition. The functions have as parameters the number of elements of the block design, the block design itself and its orbit Δ. The first (resp. second) function returns `true` if the constructed block design is a block design 1-$(n, |\Delta|, |\Delta|)$ and if it is a symmetric BIBD (resp. if it has as many blocks as elements).

Listing 6.5: Properties of the block designs in Prop. 3 according to the symmetry definitions of Magma and Key and Moori.

```
/**
 * 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 if 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 6.6 presents a validation function for Prop. 2 and 3. Magma proposes a database composed of all the primitive permutation groups of degree[3] less than or equal to $2\,500$, that is $16\,916$ groups, and $7\,643$ primitive groups of higher degree. These groups contain between 1 and $4\,095$ permutations. The Magma function `PrimitiveGroups` [BCP97] returns the sequence of these primitive groups, sorted by increasing degree, then by in-

[3]The degree of a group of permutations on a finite set Ω is the cardinality of this set Ω.

creasing cardinality (number of permutations). The code goes through this sequence and computes all the block designs from the current group thanks to the function presented in listing 6.2. For each block design, the code checks that it respects the conditions of Prop. 2 and 3, with the two interpretations of symmetry for Prop. 3. The results are saved in a file with the number of elements n, the index of the parent group in the sequence, the associated orbit Δ and the construction time of the tested block design. The calculations were performed on the computer of the Mésocentre de calcul de Franche-Comté, for the 74 first primitive groups, in $574\,107$ seconds(about 7 days), up to the degree $n = 14$ included. This is the maximal degree that can be reached within the time limit of the Mesocentre, limited to 8 days. These calculations can be reproduced freely for the first 48 primitive groups, up to degree $n = 10$, on the online calculator of Magma[4], whose use is limited to 120 seconds.

Listing 6.6: Code implementing a bounded exhaustive testing of Prop. 2 and 3.

```
/**
 * 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\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;
```

From the first 74 primitive groups of permutations, this program constructs 926 block designs, that is, all possible block designs for all primitive permutation groups of degree less than or equal to 13 and two groups of degree 14. The program found no counterexamples for Prop. 2 and 3. On the other hand, it did find 398 counterexamples for the erroneous

[4]http://magma.maths.usyd.edu.au/calc/

interpretation of Prop. 3. This is a high number for a small difference between the two definitions of symmetry. The smallest counterexample is the symmetric group S_2 with $\alpha \in \{1, 2\}$.

Example 4: *Let $S_2 = \{Id, (1, 2)\}$ be the symmetric group on $\Omega = \{1, 2\}$ and $\alpha = 1$. The stabilizer of element 1 in S_2 is the group $G_1 = \{Id\}$ and the orbits of this group are the sets $\{1\}$ and $\{2\}$. The only possible orbit Δ is the set $\{2\}$. The constructed block design is $\mathcal{B} = \{\Delta^{Id}, \Delta^{(1,2)}\} = \{\{2\}^{Id}, \{2\}^{(1,2)} = \{\{2\}, \{1\}\}$. The block design $\mathcal{B}$ is composed of 2 blocks of size 1 and 1 distinct element is found in exactly 1 block, so it is a 1-$(2, 1, 1)$ block design. It is not a 2-(v, k, λ) block design, which makes it non-symmetric according to Magma. But it is symmetric according to the definition of Key and Moori, because it has as many elements as blocks.*

6.4/ Conclusion

We have given an example of the application of programming and intensive testing to the verification of mathematical properties. We have applied bounded exhaustive testing to two propositions for the construction of block designs, according to two different interpretations, which allowed us to validate these propositions and to remove an ambiguity about their possible interpretations. In addition to documenting the code, we also identified a good practice: reducing the distance between the program and the theorem it formalizes reassures the reader on the coherence between the two.

A global perspective would be to establish and disseminate further general principles for the reproducibility and verification of mathematical publications of a computational nature.

A closer goal at the end of this work was to explore more finite geometries construction methods and to link them back to a study of contextuality. This is achieved in the following chapter.

7

Automated synthesis of contextuality proofs from subspaces of symplectic polar spaces

This chapter presents a work posted on arXiv [dHGM21b] and presented at the international conference on Quantum Physics and Logic (QPL) in 2021 [dHGM21a]. We expect to complete it further and submit it for publication in the coming months.

7.1/ Introduction

In quantum information theory, many paradoxes of the early years of quantum physics, like superposition or entanglement, have turned on to be considered as resources for the development of quantum technologies when they exhibit non-classical behavior. One of these resources is quantum contextuality. The Kochen-Specker Theorem [KS67], also proved by Bell [Bel66], is a fundamental result that establishes that no non-contextual hidden variables theories can reproduce the outcomes of quantum mechanics. First demonstrated as a mathematical result, quantum contextuality has since been tested experimentally [ARBC09, KZG^{+}09] and very recently checked on an online quantum computer [DRLB20, Hol21]. The importance of contextuality in quantum computation has been shown in [HWVE14].

The original proof of the Kochen-Specker Theorem was based on the impossibility to color bases of rays in a 3 dimensional space according to some constraints imposed by the law of quantum physics. This proof involves 117 rays. Many other proofs intending to simplify the initial argument have also been proposed [CEG96, WA11, Pla12].

In the 90's David Mermin [Mer93] and Asher Peres [Per90] introduced a different kind of argument to prove quantum contextuality. Their observable-based approach is the one that we consider in this chapter. We restate their argument, also known as the "Mermin-Peres magic square", as an example of contextual geometry, in Sect. 7.2.

The Mermin-Peres magic square and the Mermin pentagram, a three-qubit observable-based proof of the Kochen-Specker Theorem, have been investigated in the past 15 years from the perspective of finite geometry. In [HS17] it was proven by geometric arguments that these two proofs are the "smallest" ones in terms of number of observables and contexts. In [SPPH07] it was shown that the 10 possible Mermin-Peres magic squares were hyperplanes of a point-line geometry known as the doily (see details in Sect. 7.3) and in [PSH13] the number of different Mermin pentagrams was obtained and explained latter in [LHS17]. Mermin-Peres magic squares have also been considered from the perspectives of graph theory and binary constraint system games [Ark12, CM13].

In this chapter we address automatic checking of observable-based proofs of the Kochen-Specker Theorem with higher numbers of contexts and observables. In particular we check that the symplectic polar spaces W_n of rank n and order 2 are contextual for $n = 2, 3, 4$, when seen as point-line geometries encoding the commutation relations in the n-qubits Pauli group. We also prove that all hyperbolic and elliptic quadrics, which are subgeometries of W_n defined by quadratic equations, are contextual, again for $n = 2, 3, 4$. Despite the fact that elliptic and hyperbolic quadrics and their connection with multiple qubits Pauli observables have already been studied in the quantum information literature [SGL+10, LHS17], the contextuality property of those configurations has not been established before. Because those configurations involve a lot of observables and contexts (for instance 135 observables and 1575 contexts for $n = 4$), we use a computer software to check their contextuality. Looking at observable-based proofs of contextuality with large numbers of observables and contexts can be interesting to build macroscopic state-independent inequalities that violate non-contextual hidden variables inequalities [Cab08, Hol21]. Another motivation comes from quantum game theory, as more sophisticated proofs could lead to more complex game scenarios than for instance the Magic square game [BBT05].

In Sect. 7.2 we recall the perspective of finite geometry on observable-based proofs of the Kochen-Specker Theorem, and we explain how these proofs translate to the resolution of a linear system over the two-elements field $\mathbb{F}_2$. In Sect. 7.3 we recall how the symplectic polar space W_n of rank n and order 2 encodes the commutation relations in the n-qubits Pauli group, and we explain how contextual configurations of observables live in W_n as subgeometries. In Sect. 7.4 we precisely define the subgeometries characterized by quadratic equations, and we provide the contextuality results established by our program for these geometries. Sec. 7.5 is dedicated to concluding remarks. The code mentioned in this article is publicly available at https://quantcert.github.io/Magma-contextuality.

7.2/ Contextual Geometries

We first propose a precise definition of the notion of contextual geometry, based on previous work on the geometrical perspective on observable-based proofs of the Kochen-Specker Theorem. Our definition may not be as general as possible, but it is sufficient for the present work. We illustrate it with the well-known example of the Mermin-Peres magic square. Then we reformulate the contextuality property as inconsistency of a linear system with coefficients in the field of modulo-2 arithmetic.

A quantum geometry is a pair (O, C) where O is a finite set of observables (unitary finite-dimensional Hermitian operators) and C is a finite set of subsets of O, called contexts, such that

O.1 each observable $M \in O$ satisfies $M^2 = Id$ (so, its eigenvalues are in $\{-1, 1\}$);

O.2 any two observables M and N in the same context are commuting, *i.e.*, $MN = NM$;

O.3 the product of all observables in each context is either Id or $-Id$.

The elements of O and C are the points and lines of the geometry. Let the context valuation associated to (O, C) be the function $e : C \to \{-1, 1\}$ defined by $e(c) = 1$ if the product of all the observables in the context c is Id, and -1 if it is $-Id$.

A contextual geometry is a quantum geometry such that there is no (observable) valuation $f : O \to \{-1, 1\}$ such that

$$\forall c \in C, \prod_{M \in c} f(M) = e(c). \tag{7.1}$$

Otherwise, the geometry is said to be non-contextual.

7.2.1/ Example: Mermin-Peres magic square

This section presents the Mermin-Peres square completed with some notions specific to this chapter. The content of this chapter is broadly already presented in Sec. 2.4.1, but is kept here for ease of reading.

We recall that the usual Pauli matrices are

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \text{ and } Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{7.2}$$

$$\begin{tikzcd}
X \otimes I \arrow[r, dash] \arrow[d, dash] & I \otimes X \arrow[r, dash] \arrow[d, dash] & X \otimes X \arrow[d, equal] \\
I \otimes Y \arrow[r, dash] \arrow[d, dash] & Y \otimes I \arrow[r, dash] \arrow[d, dash] & Y \otimes Y \arrow[d, equal] \\
X \otimes Y \arrow[r, dash] & Y \otimes X \arrow[r, dash] & Z \otimes Z
\end{tikzcd}$$

Figure 7.1: The Mermin-Peres magic square

Let us present the Mermin-Peres magic square as a quantum geometry and restate the argument of Mermin and Peres for its contextuality. The original configuration of nine two-qubit observables proposed by Mermin [Mer93, Sect. V] is showcased in Figure 7.1. It is the quantum geometry (O, C) where $O = \{X{\otimes}I, I{\otimes}X, X{\otimes}X, I{\otimes}Y, Y{\otimes}I, Y{\otimes}Y, X{\otimes}Y, Y{\otimes}X, Z{\otimes}Z\}$ and $C = \{c_1, c_2, c_3, c_4, c_5, c_6\}$, with $c_1 = \{X \otimes I, I \otimes X, X \otimes X\}$, $c_2 = \{I \otimes Y, Y \otimes I, Y \otimes Y\}$, $c_3 = \{X \otimes Y, Y \otimes X, Z \otimes Z\}$, $c_4 = \{X \otimes I, I \otimes Y, X \otimes Y\}$, $c_5 = \{I \otimes X, Y \otimes I, Y \otimes X\}$ and $c_6 = \{X{\otimes}X, Y{\otimes}Y, Z{\otimes}Z\}$. These six sets of observables are contexts, since they contain mutually commuting observables whose product is $\pm Id$. In this example the context valuation e is such that $e(c_1) = \ldots = e(c_5) = 1$ and $e(c_6) = -1$. In Figure 7.1 the 5 positive contexts c_1 to c_5 whose product of observables is $+Id$ are depicted as simple lines, whereas the negative context c_6 whose product of observables is $-Id$ is depicted as a double line.

Since the eigenvalues of each observable are ± 1 and the measurements on each context are compatible (because the observables are mutually commuting) the product of the observed eigenvalues should be equal to the eigenvalue of the product of observables. In other words when a context is positive (resp. negative), *i.e.*, when the product of its observables is $+Id$ (resp. $-Id$), then quantum mechanics says that the product of the observed measurements should be $+1$ (resp. -1).

Now, a simple argument shows that there is no non-contextual, *i.e.* not context-dependent, deterministic function f that can assign an outcome ± 1 to each observable and satisfy at the same time the 6 constraints: if one multiplies all together the outcomes of the 6 contexts given by a non-contextual deterministic function, the result will be $+1$ because each node will show up twice in the product. However, because there is only one negative context in the Meres-Peres magic square, this product should be -1 if all constraints are satisfied.

The observable valuation f in the contextuality property (7.1) formalizes the non-contextual hidden variables hypothesis. If there were to be a deterministic non-contextual theory explaining the outcomes of quantum physics, there would be some processes in Nature, hidden from us, that would allow us to calculate these outcomes. Those hidden processes are generally called hidden variables and here f is a non-contextual function which could depend on those hidden variables: for a set of hidden variables λ, $f(M)$ is a shortcut for $f(\lambda, M)$.

7.2.2/ Contextuality as a linear problem

Let $\mathbb{F}_2 = (\{0,1\}, +, \times)$ be the two-elements field of modulo-2 arithmetic. Let (O, C) be a quantum geometry with a set $O = \{M_1, \ldots, M_p\}$ of $p = |O|$ observables/points and a set $C = \{c_1, \ldots, c_l\}$ of $l = |C|$ contexts/lines. Its incidence matrix $A \in \mathbb{F}_2^{l \times p}$ is defined by $A_{i,j} = 1$ if the i-th context c_i contains the j-th observable M_j. Otherwise, $A_{i,j} = 0$. Its valuation vector $E \in \mathbb{F}_2^l$ is defined by $E_i = 0$ if $e(c_i) = 1$ and $E_i = 1$ if $e(c_i) = -1$, where e is the context valuation of (O, C).

With these notations, the quantum geometry (O, C) is contextual *iff* the linear system

$$Ax = E \tag{7.3}$$

has no solution in $\mathbb{F}_2^p$. The matrix A being of size $l \times p$ with $l \leq p$, Eq. 7.3 can be efficiently solved with a complexity $O(p^3)$, *e.g.* by Gaussian elimination.

Finally note that A is built from the incidence structure of the quantum geometry (O, C) while the vector E comes from the signs of the contexts. In other words the left-hand side of Eq. 7.3 only depends on the geometric structure – that will be revisited in the next section – while the left-hand side depends on the outcomes predicted by quantum physics.

7.3/ Contextual configurations of the symplectic polar space

We aim to automate the generation of quantum geometries and the detection of their contextuality, with observables restricted to the elements of the n-qubits Pauli group P_n, *i.e.* the group of the n-fold tensor products of Pauli matrices. We study them through their encoding as vectors from a vector space over the two-elements field $\mathbb{F}_2$. This encoding does not preserve all information, in particular we loose the commutation relations and the signs of the contexts. At the geometrical level, the commutation relation will be recovered by introducing a nondegenerate symplectic form (Sect. 7.3.1). The signs of the contexts will be determined as detailed in Sect. 7.3.2.

7.3.1/ THE SYMPLECTIC POLAR SPACE W_n

Let $V_n = \mathbb{F}_2^{2n}$ be the vector space of dimension $2n$ over the two-elements field $\mathbb{F}_2$. Let $\bigotimes$ denote the (generalized) tensor product. An element of the n-qubits Pauli group P_n is an operator O which can be factorized as $O = s \bigotimes_{i\in[1..n]} Z^{a_i}X^{b_i}$ with $a_i, b_i \in \mathbb{F}_2$ and a phase $s \in \{\pm 1, \pm i\}$. We denote by C_n the center of P_n, *i.e.* $C_n = \{\pm I, \pm iI\}$. The surjective map $\pi : P_n \to V_n$ defined by $\pi(s \bigotimes_{i\in[1..n]} Z^{a_i}X^{b_i}) = (a_1, b_1, \dots, a_n, b_n)$ factors to the isomorphism $\overline{\pi} : P_n/C_n \to V_n$ such that $\overline{\pi}(\overline{O}) = (a_1, b_1, \dots, a_n, b_n)$, where $\overline{O}$ is the class of O in P_n/C_n [Hol21, Sect. 2].

The map $\overline{\pi}$ has several crucial aspects: its images are more elementary than the original objects (binary vectors replace Hermitian matrices), and $\overline{\pi}$ preserves some key properties about P_n. As defined, $\overline{\pi}$ already transforms the matrix product into the sum in V_n. In order to encode commutation relation, we define the inner product (also called symplectic form) on V_n as $\langle x|y\rangle = xJy^\intercal$, with

$$J = \begin{pmatrix} \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} & & \\ & \ddots & \\ & & \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} \end{pmatrix}.$$

One can also define the inner product by developing the previous formula: for two vectors $x = (a_1, b_1, \dots, a_n, b_n)$ and $y = (a'_1, b'_1, \dots, a'_n, b'_n)$, their inner product is defined as $\langle x|y\rangle = \sum_{i=1}^{n} a_i b'_i + a'_i b_i$.

Using this inner product, we have $\left\langle \overline{\pi}(\overline{O}) \middle| \overline{\pi}(\overline{O'}) \right\rangle = 0$ *iff* the operators in $\overline{O}$ and $\overline{O'}$ commute [Hol21, Sect. 2].

Since the trivial n-qubits Pauli operator I does not correspond to a measurement, we eliminate its class $\overline{I}$ from P_n/C_n and the corresponding neutral element $\overline{\pi}(\overline{I}) = (0, \dots, 0)$ from V_n. This restriction of $\overline{\pi}$ is a bijection between the set of non trivial n-qubits Pauli observables and the projective space $PG(2n-1, 2)$, whose points are nonzero vectors in V_n.

We can now define a counterpart to a quantum geometry in $PG(2n-1, 2)$. A quantum configuration is a pair (P, L) where P is a finite set of points of $PG(2n-1, 2)$ and L is a finite set of subsets of P, such that

S.1 any two vectors V and W in the same element of L are commuting, *i.e.* $\langle V|W\rangle = 0$;

S.2 the sum of all vectors in each element of L is $(0, \dots, 0)$.

One can see that this definition of a quantum configuration corresponds through $\overline{\pi}$ to that of a quantum geometry given in Sect. 7.2. Indeed condition **O.1** is satisfied by the elements of P_n, so the fact that we use only points from $PG(2n-1, 2)$ satisfies it. Condition **O.2** is in correspondence with Condition **S.1** and Condition **O.3** is in correspondence with Condition **S.2**.

A totally isotropic subspace of $PG(2n-1, 2)$ is a linear subspace S of $PG(2n-1, 2)$ such that $\langle a|b\rangle = 0$ for any $a, b \in S$. The name comes from the quadratic forms which are said to be isotropic when they are annihilated on a non null point. Thus, Condition **S.1** rewrites as "all elements of L are totally isotropic subspaces". The space of totally isotropic subspaces of $PG(2n-1, 2)$ for $\langle|\rangle$ is called the symplectic polar space of rank n and order 2 and is denoted by $W(2n-1, 2)$ (abbreviated as W_n). The symplectic space W_n is a set of subspaces of different dimensions: the most elementary ones are the points of W_n, they are the totally isotropic subspaces of dimension 0. In all rigor they are the singletons $\{v\}$, for all points $v \in PG(2n-1, 2)$, but we identify them with their element v. They do not satisfy Condition **S. 2**, whereas all other totally isotropic subspaces (of positive dimension) satisfy it.

In this work, we only consider the pairs (P, L) such that P is a set of points in W_n and L is a subset of W_n composed of totally isotropic subspaces with the same positive dimension. By construction, such a (P, L) satisfies Conditions **S.1** and **S.2**, so it is a quantum configuration.

Example 5: *The symplectic polar space of rank 2 and order 2, $W_2 = W(3, 2)$, corresponds to Pauli operators acting on two qubits. It has 15 points (the subspaces of dimension 0) and 15 lines (the subspaces of dimension 1), and no subspace of higher dimension. It is represented in Fig. 7.2 as a finite geometry, where each point of the symplectic space is a point of the geometry and the three elements of each isotropic subspace of dimension 1 are shown as a line.*

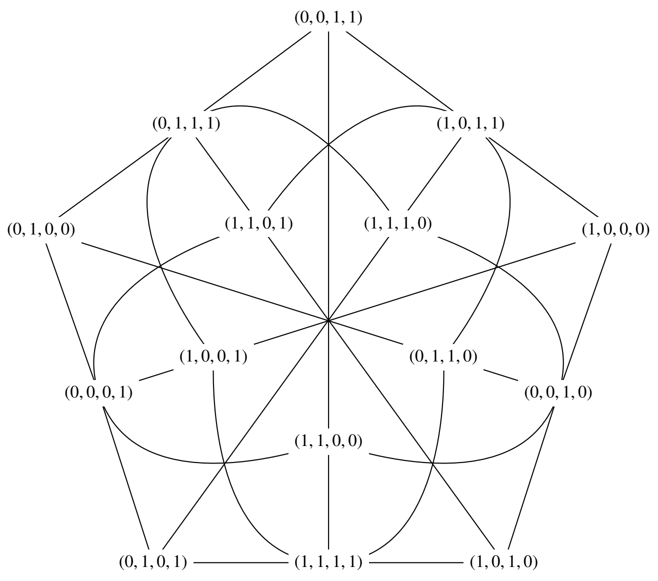


Figure 7.2: The Doily, a point-line representation of $W_2 = W(3, 2)$

Notice an importance difference compared to traditional geometry: two points are not necessarily aligned in finite geometry. For instance, the points $(0, 1, 1, 1)$ and $(1, 0, 1, 1)$ are not on the same line since $\langle(0, 1, 1, 1)|(1, 0, 1, 1)\rangle = 1 \neq 0$.

In W_2, the pair (P, L) may have the following content:

$$\begin{aligned} P = \{&(0,1,0,0),(0,0,0,1),(0,1,0,1),\\ &(1,1,0,0),(0,0,1,1),(1,1,1,1),\\ &(1,1,0,1),(0,1,1,1),(1,0,1,0)\} \end{aligned}$$

and

$$\begin{aligned} L = \{&\{(0,1,0,0),(0,0,0,1),(0,1,0,1)\},\\ &\{(1,1,0,0),(0,0,1,1),(1,1,1,1)\},\\ &\{(1,1,0,1),(0,1,1,1),(1,0,1,0)\},\\ &\{(0,0,0,1),(1,1,0,0),(1,1,0,1)\},\\ &\{(0,1,0,0),(0,0,1,1),(0,1,1,1)\},\\ &\{(0,1,0,1),(1,1,1,1),(1,0,1,0)\}\} \end{aligned}$$

This is in fact the Mermin-Peres square, presented in Fig. 7.1

Example 6: *Another example of symplectic space may be the even more elementary W_1. It contains only three points, $(0,1)$, $(1,0)$ and $(1,1)$ and no subspaces of positive dimension.*

7.3.2/ Context valuation, valuation vector and contextual configurations

Any quantum configuration (P, L) in W_n with $p = |P|$ points and $l = |L|$ (context) lines determines an incidence matrix $A \in \mathbb{F}_2^{l\times p}$ defined by $A_{i,j} = 1$ if the i-th element of L contains the j-th point in P. However, it provides no context/line valuation e, on which its contextuality however depends. A context valuation can be derived as follows from an interpretation of points in W_n as Pauli operators, in other words from a right inverse ρ of the map π ($\pi \circ \rho = id$). Among all these inverses, we consider here the map ρ : $PG(2n-1, 2) \to P_n$ defined by $\rho((a_1, b_1, \ldots, a_n, b_n)) = \bigotimes_k O_k$ with $O_k = (-i)^{a_k b_k} Z^{a_k} X^{b_k}$. One can develop this expression as

$$O_k = \begin{cases} I \text{ if } (a_k, b_k) = (0,0) \\ X \text{ if } (a_k, b_k) = (0,1) \\ Y \text{ if } (a_k, b_k) = (1,1) \\ Z \text{ if } (a_k, b_k) = (1,0) \end{cases}.$$

The corresponding context valuation is the map $e_\rho : L \to \{-1, 1\}$ such that $\prod_{p \in l} \rho(p) = e_\rho(l)\ Id$ for all $l \in L$. It results from the commutation relations on each context line that the values of $e_\rho(l)$ can only be ± 1. The corresponding valuation vector E_ρ is defined from e_ρ as in Sect. 7.2.2. Finally, a contextual configuration is a triple (P, L, ρ) composed of a quantum configuration (P, L) and an interpretation ρ such that the linear system $Ax = E_\rho$ has no solution in $\mathbb{F}_2^p$ (in this definition, A is the incidence matrix of (P, L)).

Example 7: *After replacing each point in W_2 by its image by ρ, Fig. 7.2 becomes Fig. 7.3. The product of all observables on each line is I (marked as a single black line) or $-I$ (marked as a doubled red line). It determines the values 1 and -1 of e_ρ. With these elements, it is well-known that the Doily is a contextual configuration (see e.g. [Cab08]).*

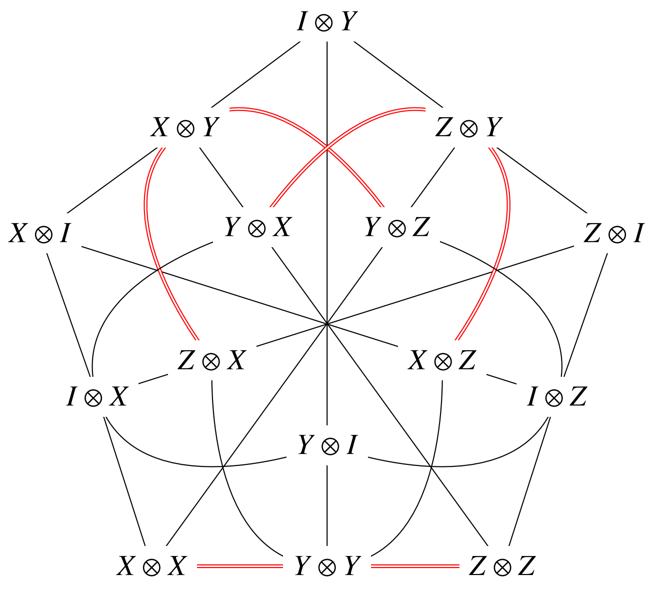


Figure 7.3: The Doily with Pauli operators

Notice that the Mermin-Peres square shown in Fig. 7.1 is a subgeometry of the Doily, as shown in Fig. 7.4.

7.4/ Contextual Subspaces of Symplectic Polar Spaces

In order to automate the detection of a large number of contextual configurations, we first fix some programmable criteria to characterize the candidate geometries. We first define these geometries. Then we sum up the results obtained.

A line of W_n is a totally isotropic subspace of W_n of dimension 1. An element of W_n of maximal dimension $(n-1)$ is called a generator. A quadric of W_n is the set of points that annihilate a given quadratic form. The most elementary quadratic form that we could define is $Q_0(x) = x_1x_2 + \ldots + x_{2n-1}x_{2n}$ if $x = (x_1, x_2, \ldots, x_{2n})$. A form is an homogeneous

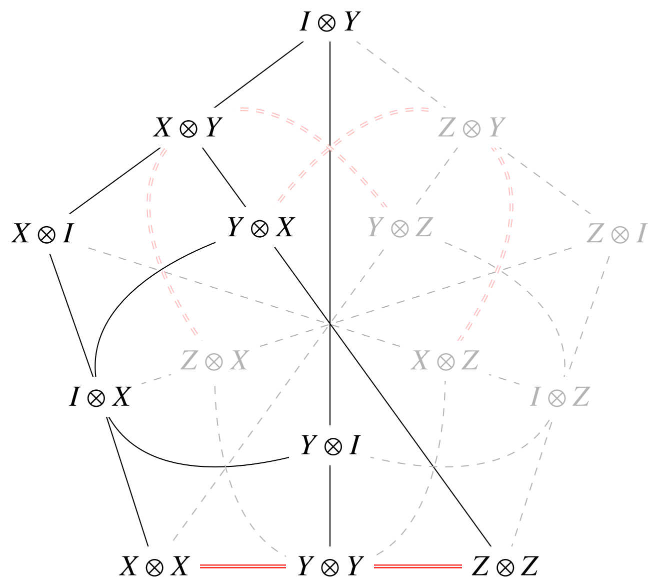


Figure 7.4: The Mermin-Peres square as a subgeometry of the Doily

polynomial – a sum of monomial of same degree – and it is a quadratic form if the degree of the monomials is 2. An hyperbolic quadratic form is a form Q_p defined by $Q_p(x) = Q_0(x) + \langle x|p\rangle$ where $p \in V_n$ annihilates Q_0, *i.e.* when $Q_0(p) = 0$, whereas an elliptic form is such a form Q_p where p does not annihilate Q_0. Finally, an hyperbolic (resp. elliptic) quadric symplectic polar space is a quadric corresponding to the zero locus of an hyperbolic (resp. elliptic) quadratic form. The perpset of the point $v \in W_n$ is the set of all points w isotropic to v, *i.e.* such that $\langle v|w\rangle = 0$.

Remark 7: *A (geometric) hyperplane is a set of points in W_n such that any line of W_n either has a single point of intersection with the hyperplane, or is fully contained in it. The 10 Mermin-Peres squares of W_2 are geometric hyperplanes. It is also known that the elliptic quadrics, the hyperbolic quadrics and the perpsets are the only types of hyperplanes in W_n [VL10]. In fact the 10 Mermin-Peres squares are the 10 hyperbolic quadrics of W_2. This remark motivates our choice to study the contextuality of all types of hyperplanes in W_n.*

Given these definitions, we consider the following families of geometries:

G.1 all points and all lines of W_n;

G.2 all points and all generators of W_n;

G.3 all points and all lines in one hyperbolic quadric in W_n, for all hyperbolic quadrics in W_n;

G.4 all points and all lines in one elliptic quadric in W_n, for all elliptic quadrics in W_n;

G.5 all points and all lines in one perpset in W_n, for all perpsets of a given point of W_n.

Example 8: *For W_2, the single element of the family* **G.1** *is represented by the Doily (Fig. 7.2): the set of points P is the set of points of W_2 and the set of lines L is the set of lines represented in the Doily. There is always a single geometry in this family.*

In the 2 qubits case, the lines are the same as the generators (the lines are the subspaces of maximal dimension). This is not the case for $n > 2$ though. This means that the single element of the family **G.2** *is also represented by the Doily in the two qubits case.*

In the 2 qubits case, the Mermin-Peres square is an example of hyperbolic quadric, this means that Fig. 7.4 is an example of element of the family **G.3** *(this particular quadric is generated by the point $X \otimes X$).*

Until now, I have not shown any example of elliptic quadric. To do so, let's take a point that does not annihilate Q_0, for instance $p = Z \otimes X$. The set of points that annihilate Q_p is $P = \{X \otimes I, Y \otimes X, Y \otimes Z, Z \otimes I, Y \otimes Y\}$. The corresponding element of the family **G.4** *is the geometry (P, L) where P is given above and L is the set of all lines such as each line is both a line of W_n and a subset of P. This set L is always empty for 2 qubits, which explains the N/A in table 7.2.*

Finally, the family **G.5** *is the set of all perpsets of W_n, which means that we can exhibit an example by taking a specific perpset. Let us consider the perpset generated by $Y \otimes X$, if is composed of all points non collinear with $Y \otimes X$, i.e. $P = \{I \otimes X, Z \otimes Y, Y \otimes X, Y \otimes I, X \otimes Z, X \otimes Y, Z \otimes Z\}$. As is previous examples, we are looking for lines of W_2 which are subsets of P, this gives us $L = \{\{I \otimes X, Y \otimes X, Y \otimes I\}, \{Y \otimes X, X \otimes Y, Z \otimes Z\}, \{Z \otimes Y, Y \otimes X, X \otimes Z\}\}$.*

7.4.1/ Results

Our contextual configurations detection program is implemented in the language of the well-established tool Magma for theoretical mathematics. In particular, Magma provides us with $\mathbb{F}_2$ and $PG(2n-1, 2)$ equipped with the symplectic form. From this, our program generates the incidence structure of the generators of W_n – using the commutation relations of the points of W_n viewed as a graph and the integrated Magma function for buildings incidence structures from graphs – and uses it to compute all totally isotropic subspaces of dimension 1 with an in-house algorithm. For **G.1**, **G.3**, **G.4** and **G. 5** the totally isotropic subspaces of dimension 1 are scanned and possibly filtered according to the characteristics of each geometry, to build an incidence matrix. For **G.2** the totally isotropic subspaces of dimension $(n-1)$ and the incidence matrix are built on the fly. For all these geometries, we compute their line valuation, thanks to an implementation of ρ. Then, contextuality is detected by a call to the Magma function `IsConsistent` which determines whether a linear system has a solution. The code ran on the supercomputer of the Mésocentre de calcul de Franche-Comté.

For 2 qubits, the computation takes less than 0.1s, for 3 qubits the computation takes around 5s. But for 4 qubits, the computation already takes around 10min, and for 5 qubits, the computation takes around 24h.

These durations are consistent with the algorithmic complexity of the functions computing the families of geometries, presented in Tab. 7.1. These complexities are calculated as follows: the symplectic space contains 2^{2n} points, so iterating over it takes $O(2^{2n})$ steps. Each line contains three points, the third one being the sum of the other two, so the complexity to generate all lines (for **G.1**) is $O\left(2^{2n} \times 2^{2n}\right) = O\left(2^{4n}\right)$. This is consistent with the number $(4^n - 1)(4^{n-1} - 1)/3$ of lines in the symplectic space. To compute the set of all quadrics, we iterate over all the points. For each point we generate its quadratic form and we iterate over all the points to find those who annihilate this quadratic form. These points are the points of the quadric. The complexity of these two operations is negligible compared to that of the next one: iteration over all the lines of the symplectic space, a line being selected if it is a subset of the points of the quadric. This yields a complexity in $O\left(2^{2n} \times 2^{4n}\right) = O\left(2^{6n}\right)$. The computation for the perpsets is very similar. For the family **G.2** of generators, we use the property that they are the blocks of the incidence structure whose elements are the points of the symplectic space and such that two points are incident if and only if they commute. The most expensive operation is the generation of the $\prod_{i\in[1..n]}(2^i+1)$ blocks of this incidence structure, resulting in a complexity in $O\left(2^{n(n+1)/2}\right)$.

Geometries	Complexity
Lines (**G.1**)	$O\left(2^{4n}\right)$
Generators (**G.2**)	$O\left(2^{n(n+1)/2}\right)$
Quadrics and Perpsets (**G.3** + **G.4** + **G.5**)	$O\left(2^{6n}\right)$

Table 7.1: Algorithmic complexity for each geometry family

Table 7.2 presents the contextuality results in a table. Each entry yields the contextuality as well as the number of elements in the family. Note that the elliptic quadrics for $n = 2$, also known as ovoids, do not contain any line. Therefore there are no contexts in this case and that is why we indicate here "N/A". We also could have skipped the computation of the generators for $n = 2$, because their dimension is $n - 1 = 1$, so they are lines, already computed for (**G.1**). We kept it as it is reassuring to see that we indeed obtain the same result for both families.

The cells in **bold font** represent the results that were not previously known. In each cell, "C" means that the geometries are contextual and "N" means they are not.

The number of objects in each class was previously known, [VL10] gives a good overview of these numbers, which we recall and complete in Tab. 7.3 for convenience.

Remark 8: *The contextuality of the configurations (**G.1**) has been established in [Cab08]. We notice that the contextual nature of a given configuration remains the same among all*

Geometries	$n = 2$	$n = 3$	$n = 4$	$n = 5$
Lines (**G.1**)	C(1)	C(1)	C(1)	C(1)
Generators (**G.2**)	C(1)	**N(1)**	**N(1)**	**N(1)**
Hyperbolics (**G.3**)	C(10)	**C(36)**	**C(136)**	**C(528)**
Elliptics (**G.4**)	N/A (6)	**C(28)**	**C(120)**	**C(496)**
Perpsets (**G.5**)	N(15)	**N(63)**	**N(255)**	**N(1023)**

Table 7.2: Contextuality results

Geometries (P, L)	Cardinality of P	Cardinality of L	Number of geometries
Lines (**G.1**)	$4^n - 1$	$\frac{(4^n-1)(4^{n-1}-1)}{3}$	1
Generators (**G.2**)	$2^{n-1} - 1$	$\prod_{i\in[1..n]}(2^i + 1)$	1
Hyperbolics (**G.3**)	$\frac{4^n+2^n}{2} - 1$	$(\frac{4^n+2^n}{2} - 1)(\frac{4^{n-1}+2^{n-1}}{2} - 1)/3$	$\frac{4^n+2^n}{2}$
Elliptics (**G.4**)	$\frac{4^n-2^n}{2} - 1$	$(\frac{4^n-2^n}{2} - 1)(\frac{4^{n-1}-2^{n-1}}{2} - 1)/3$	$\frac{4^n-2^n}{2}$
Perpsets (**G.5**)	$\frac{4^n}{2} - 1$	$4^{n-1} - 1$	$4^n - 1$

Table 7.3: Cardinalities for each geometry family and their members

the geometries in the same family, and for all sizes. For instance all hyperbolic quadrics are contextual. The only exception is **G.2** *where this is not the case for $n = 2$, but as explained earlier, this comes from the fact that, in this case, generators are in fact lines. From the geometry construction it is clear that for a fixed n, the matrix A in Eq. 7.3 is the same (up to a change of basis) for all geometries in the same family (it can also be seen from the fact that the symplectic group Sp$(2n, 2)$ acts transitively on the set of geometric hyperplanes [VL10]). However the vector E of the right-hand side of Eq. 7.3 is not the same for all hyperbolic quadrics. To get a better understanding of this, it will be interesting to understand for instance how the symplectic group Sp$(2n, 2)$ acts on the labeling of the contexts of W_n.*

7.5/ Conclusion

To sum up, Sect. 7.2 introduced observable-based contextuality proofs and a way to detect them by solving a single linear system per proof. Sect. 7.3 showed how to generate these proofs using a simpler representation of the observables and how to translate these proofs into objects from Sect. 7.2. This allowed us to generate entire families of proofs in Sect. 7.4.

Doing this, we have performed a systematic study of observable-based contextuality proofs with larger numbers of observables than in previous work. This systematic approach brought us several research directions for future work. First, we have now several conjectures that we would like to further investigate, *e.g.* interpretation (choice of ρ) invariance of given results, and scale invariance of contextuality for a given family. We also intend to perform formal proofs of theorems about observable-based proofs of the Kochen-Specker Theorem, based on the formal definitions of the notions of contextual geometry and contextual configuration proposed in the present chapter.

This study also raised interesting questions about the structure of the geometries sets. Indeed, if some families of geometries are contextual and others are not, it would be interesting to discover an underlying structure justifying this fact. This lead us to the work presented in the following chapter.

8

Taxonomy of Polar Subspaces of Multi-Qubit Symplectic Polar Spaces of Small Rank

This chapter presents parts of a published article [SdHG21] closely following the content of Chap. 7. Various notions are redefined in this chapter, to make it self-contained. The reader might be surprised to see that some notions are defined differently, this is the case because different definitions fit better different needs, this is for example the case for the symplectic form. But the various definitions are in the end equivalent. Another notation difference is kept to fit the original paper: a quantum configuration (P, L) is the same as a point-line incidence geometry $\Gamma(P, L)$. I did not include some elements (most notably section 5) of the published article because I mostly participated in the early sections. Furthermore, the latter sections of the article, although very interesting, do not add new material considering the big picture of this manuscript: they do not seem to add new possibilities of formal proof of protocol contextuality.

8.1/ Introduction

Some fifteen years ago it was discovered (see, e.g., [SP07, PS08, HOS09, Tha09]) that there exists a deep connection between the structure of the n-qubit Pauli group and that of the binary symplectic polar space of rank n, $W(2n-1, 2)$, where commutation relations between elements of the group are encoded in collinearity relations between points of $W(2n-1, 2)$. This connection has subsequently been used to get a deeper insight into, for example, finite geometric nature of observable-based proofs of quantum contextuality (for a recent review, see [Hol19a]), properties of certain black-hole entropy formulas [LSVP09] and the so-called black-hole/qubit correspondence [BDL12], leading to finite-geometric underpinning four distinct Hitchin's invariants and the Cartan invariant of form theories of gravity [LHS17] and even to an intriguing finite-geometric toy model of space-time [LH19]. This group-geometric connection was further strengthened by making use of the concept of geometric hyperplane and that of the Veldkamp space of $W(2n-1,2)$ [VL10]. As per quantum contextuality, famous two-qubit Mermin-Peres magic squares were found to be isomorphic to a special class of geometric hyperplanes of $W(3,2)$ called grids [SPPH07], whereas three-qubit Mermin pentagrams were found to have their natural settings in the magic Veldkamp line of $W(5,2)$ [LS17], being also isomorphic – under the grassmannian correspondence of type Gr$(2,4)$ – to ovoids of $W(3,2)$ [SL12]. Concerning the black-hole/qubit correspondence, here a key role is played by the geometric hyperplane isomorphic to an elliptic quadric of $W(5,2)$. Interestingly, form theories of gravity seem to indicate that also a certain part of the magic Veldkamp line in the four-qubit symplectic polar space, $W(7,2)$, and the associated extended geometric hyperplanes are of physical relevance.

From the preceding paragraph it is obvious that revealing finer traits of the structure of binary symplectic polar spaces of small rank can be vital for further physical applications of these spaces. Having this in view, we will focus on sets of $W(2n-3,2)$'s located in $W(2n-1,2)$, for $n = 2,3,4$, providing their comprehensive observable-based taxonomy. Key parameters of our classification of such subspaces of $W(2n-1,2)$ will be: the number of negative lines they contain (which is also an important parameter when it comes to quantum contextuality), the distribution of different types of observables they feature, the character of the geometric hyperplane a subspace of a given type shares with the distinguished (non-singular) quadric of $W(2n-1,2)$ and, in the case of refined 'decomposition' of three-qubit $W(3,2)$'s, also the very structure of their Veldkamp lines.

The chapter is organized as follows. Sec. 8.2 provides the reader with the necessary finite-geometric background and notation. Sec. 8.3 deals with $W(3,2)$ and the hierarchy of its triads, Sec. 8.4 – the central subsection of the chapter – addresses the three-qubit $W(5,2)$ and its doilies and Sec. 8.5 classifies $W(5,2)$'s living in the four-qubit $W(7,2)$. Finally, Sec. 8.6 is devoted to concluding remarks.

8.2/ Finite Geometry Background

Given a d-dimensional projective space $PG(d,2)$ over $\mathbb{F}_2$, a polar space $\mathcal{P}$ in this projective space consists of the projective subspaces that are *totally isotropic/singular* with respect to a given non-singular bilinear form; $PG(d,2)$ is called the ambient projective space of $\mathcal{P}$. A projective subspace of maximal dimension in $\mathcal{P}$ is called a generator; all generators have the same (projective) dimension $r-1$. One calls r the rank of the polar space.

Polar spaces of relevance for us are of three types (see, for example, [HT16, Cam92]):

symplectic, hyperbolic and elliptic. Recall (Chap. 7) the symplectic polar space $W(2n-1,2)$, for $n \geq 1$, consists of all the points of $PG(2n-1,2)$, $\{(x_1,x_2,\ldots,x_{2n}) : x_j \in \{0,1\}, j \in \{1,2,\ldots,2n\}\}\backslash\{(0,\ldots,0)\}$, together with the totally isotropic subspaces with respect to the standard symplectic form

$$\sigma(x,y) = x_1y_{n+1} - x_{n+1}y_1 + x_2y_{n+2} - x_{n+2}y_2 + \cdots + x_ny_{2n} - x_{2n}y_n. \tag{8.1}$$

This space features

$$|W|_p = 4^n - 1 \tag{8.2}$$

points and

$$|W|_g = (2+1)(2^2+1)\cdots(2^n+1) \tag{8.3}$$

generators. The standard hyperbolic orthogonal polar space (sometimes referred simply as hyperbolic space) $Q^+(2n-1,2)$, for $n \geq 1$, is formed by all the subspaces of $PG(2n-1,2)$ that lie on a given non-singular hyperbolic quadric, with the standard equation

$$x_1x_{n+1} + x_2x_{n+2} \ldots + x_nx_{2n} = 0. \tag{8.4}$$

The standard hyperbolic space as well as each hyperbolic space obtained by symplectic transformation (variable change of the coordinates maintaining the standard equation) of the standard hyperbolic space contains

$$|Q^+|_p = (2^{n-1}+1)(2^n-1) \tag{8.5}$$

points and there are

$$|W|_{Q^+} = |Q^+|_p + 1 = (2^{n-1}+1)(2^n-1) + 1 \tag{8.6}$$

copies of them in $W(2n-1,2)$. Finally, the standard elliptic orthogonal polar space $Q^-(2n-1,2)$, for $n \geq 2$, comprises all points and subspaces of $PG(2n-1,2)$ satisfying the standard equation

$$(x_1x_{n+1} + x_1^2 + x_{n+1}^2) + x_2x_{n+2} + \cdots + x_nx_{2n} = 0, \tag{8.7}$$

$XY + X^2 + Y^2$ being the only irreducible polynomial of degree 2 over $\mathbb{F}_2$. The standard elliptic space as well as each elliptic space obtained by symplectic transformation (variable change of the coordinates maintaining the standard equation) of the standard elliptic space contains

$$|Q^-|_p = (2^{n-1}-1)(2^n+1) \tag{8.8}$$

points and there are

$$|W|_{Q^-} = |Q^-|_p + 1 = (2^{n-1}-1)(2^n+1) + 1 \tag{8.9}$$

copies of them in $W(2n-1,2)$. For both symplectic and hyperbolic polar spaces $r = n$, whereas for the elliptic one $r = n-1$. The smallest non-trivial symplectic polar space is the $n = 2$ one, $W(3,2)$, often referred to as the *doily*. It features 15 points (see Eq. 8.2) and the same number of lines (that are also its generators, see Eq. 8.3), with three points per line and three lines through a point; it is a self-dual 15_3-configuration and the only one out of 245 342 such configurations that is triangle-free, being, in fact, isomorphic to the generalized quadrangle of order two (GQ$(2,2)$). This symplectic polar space features ten $Q^+(3,2)$'s (by Eq. 8.6) and six $Q^-(3,2)$'s (by Eq. 8.9). A $Q^+(3,2)$ contains nine points and six lines forming a 3×3 grid, so it is also called a grid. A $Q^-(3,2)$ features five pairwise non-collinear points, hence it is an ovoid. A triple of mutually non-collinear points of $W(3,2)$ is called a *triad* and a point collinear with all the three points of a triad is called a *center* of the triad; $W(3,2)$ contains 60 unicentric and 20 tricentric triads.

The n-qubit observables we will be dealing with belong to the set

$$\mathcal{S}_n = \{G_1 \otimes G_2 \otimes \cdots \otimes G_n : \ G_j \in \{I, X, Y, Z\}, \ j \in \{1, 2, \ldots, n\}\} \backslash \{\mathcal{I}_n\} \tag{8.10}$$

where $\mathcal{I}_n \equiv I_2 \otimes \ldots \otimes I_2$, X, Y and Z are the Pauli matrices, I is the identity matrix and '$\otimes$' stands for the tensor product of matrices. $\mathcal{S}_n$, whose elements are simply those of the n-qubit Pauli group if the global phase is disregarded, features two kinds of observables, namely *symmetric* (i.e., observables featuring an even number of Y's) and *skew-symmetric*; the number of symmetric observables is $(2^{n-1}+1)(2^n-1)$. We shall further employ a finer classification where an observable having $n-1, n-2, n-3, \ldots I$'s will be, respectively, of type A, B, C, $\ldots$; also, whenever it is clear from the context, $G_1 \otimes G_2 \otimes \cdots \otimes G_n$ will be short-handed to $G_1 G_2 \cdots G_n$.

For a particular value of n, the $4^n - 1$ elements of $\mathcal{S}_n$ can be bijectively identified with the same number of points of $W(2n-1,2)$ in such a way that the images of two commuting elements lie on the same line of this polar space, and *generators* of $W(2n-1,2)$ correspond to maximal sets of mutually commuting elements. If we take the symplectic form defined by Eq. 8.1, then this bijection acquires the form

$$G_j \leftrightarrow (x_j, x_{j+n}), \ j \in \{1, 2, \ldots, n\}, \tag{8.11}$$

assuming that

$$I \leftrightarrow (0,0), \ X \leftrightarrow (0,1), \ Y \leftrightarrow (1,1), \text{ and } Z \leftrightarrow (1,0). \tag{8.12}$$

Employing the above-introduced bijection (for more details see, e. g., [LS17]), it can be shown that given an observable O, the set of symmetric observables commuting with O together with the set of skew-symmetric observables not commuting with O will lie on a certain non-degenerate quadric of $W(2n-1,2)$, this quadric being hyperbolic (resp. elliptic) if O is symmetric (resp. skew-symmetric). We can express this important property by making, whenever appropriate, this associated observable explicit in a subscript, $Q^{\pm}_{(O)}(2n-$

$1, 2)$, note that there exists a particular hyperbolic quadric associated with $\mathcal{I}$:

$$Q^{+}_{(\mathcal{I})}(2n-1,2) := \{(x_1, x_2, \ldots, x_{2n}) \in W(2n-1,2) \mid x_1x_{n+1} + x_2x_{n+2} + \ldots + x_nx_{2n} = 0\}. \tag{8.13}$$

This definition is a qualitative one, but it can also be fully formalized, as it was in Chap. 7. Another way to describe $Q^{\pm}_{(O)}(2n-1,2)$ can be given as such: for a single qubit, $Q_{(O)}(x_1, x_2) = x_1x_2 + ix_1^2 + jx_2^2$ where $O \to (i, j)$ is given in Eq. 8.12. With this, we define for any operator $O_1O_2 \ldots O_n$

$$Q^{\pm}_{(O_1O_2\ldots O_n)}(2n-1,2) = \{(x_1, x_2, \ldots, x_{2n}) \in W(2n-1,2) \mid Q_{(O_1)}(x_1, x_{n+1}) + Q_{(O_2)}(x_2, x_{n+2}) + \ldots + Q_{(O_n)}(x_n, x_{2n}) = 0\}.$$

Given a point-line incidence geometry $\Gamma(P, L)$, a *geometric hyperplane* of $\Gamma(P, L)$ is a subset of its point set such that a line of the geometry is either *fully* contained in the subset or has with it just a *single* point in common. The Veldkamp space $\mathcal{V}(\Gamma)$ of $\Gamma(P, L)$ is the space in which [BC13]:

(i) a point is a geometric hyperplane of Γ and

(ii) a line is the collection, denoted $H'H''$, of all geometric hyperplanes H of Γ such that $H' \cap H'' = H' \cap H = H'' \cap H$ or $H = H', H''$, where H' and H'' are distinct points of $\mathcal{V}(\Gamma)$.

For a $\Gamma(P, L)$ with three points on a line, all Veldkamp lines are of the form $\{H', H'', \overline{H'\Delta H''}\}$ where $\overline{H'\Delta H''}$ is the complement of symmetric difference of H' and H'', *i.e.* they form a vector space over $\mathbb{F}_2$. As demonstrated in [VL10], $\mathcal{V}(W(2n-1,2)) \cong PG(2n, 2)$. Its points are both hyperbolic and elliptic quadrics of $W(2n-1,2)$, as well as its perpsets. Given a point x of $W(2n-1,2)$, the *perpset* $\widehat{Q}_{(x)}(2n-1,2)$ of x consists of all the points collinear with it,

$$\widehat{Q}_{(x)}(2n-1,2) := \{y \in W(2n-1,2) \mid \sigma(x, y) = 0\};$$

the point x being referred to as the *nucleus* of $\widehat{Q}_{(x)}(2n-1,2)$.

Example 9: *[Hol19b] Let us take the Veldkamp space of the Mermin-Peres square as an example. First of all, some of the hyperplanes are highlighted in red in Fig. 8.1.*

With this, an example of line of the corresponding Veldkamp space is shown in Fig. 8.2. As stated previously, this set is indeed of the shape $\{H', H'', \overline{H'\Delta H''}\}$.

We shall briefly recall basic properties of the Veldkamp space of the doily, $\mathcal{V}(W(3,2)) \simeq PG(4,2)$, whose in-depth description can be found in [SPPH07]. The 31 points of $\mathcal{V}(W(3,2))$ comprise 15 perpsets, ten grids and six ovoids – as also illustrated in Fig. 8.3 that depicts the three kinds of geometric hyperplanes of $W(3,2)$. The 15 points of the doily are represented by small circles and its 15 lines are illustrated by the straight

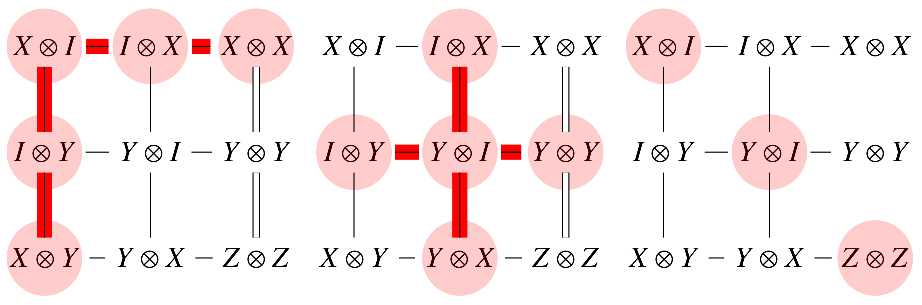

$X\otimes I$ — $I\otimes X$ — $X\otimes X$
$I\otimes Y$ — $Y\otimes I$ — $Y\otimes Y$
$X\otimes Y$ — $Y\otimes X$ — $Z\otimes Z$

Figure 8.1: Representation of some hyperplanes of the Mermin-Peres square.

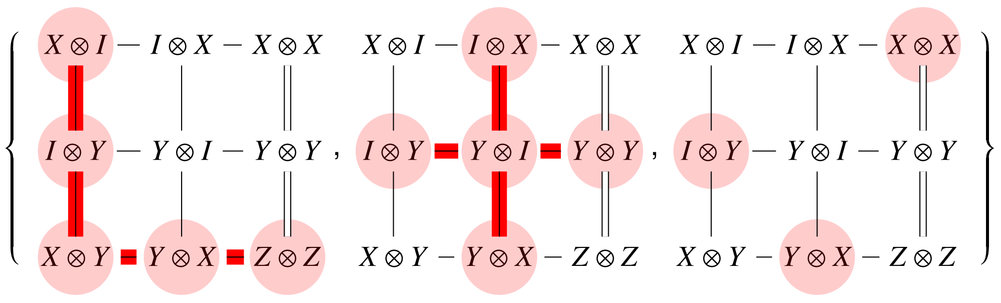

$X\otimes I$ — $I\otimes X$ — $X\otimes X$
$I\otimes Y$ — $Y\otimes I$ — $Y\otimes Y$
$X\otimes Y$ — $Y\otimes X$ — $Z\otimes Z$

Figure 8.2: One line of the Veldkamp space of the Mermin-Peres square.

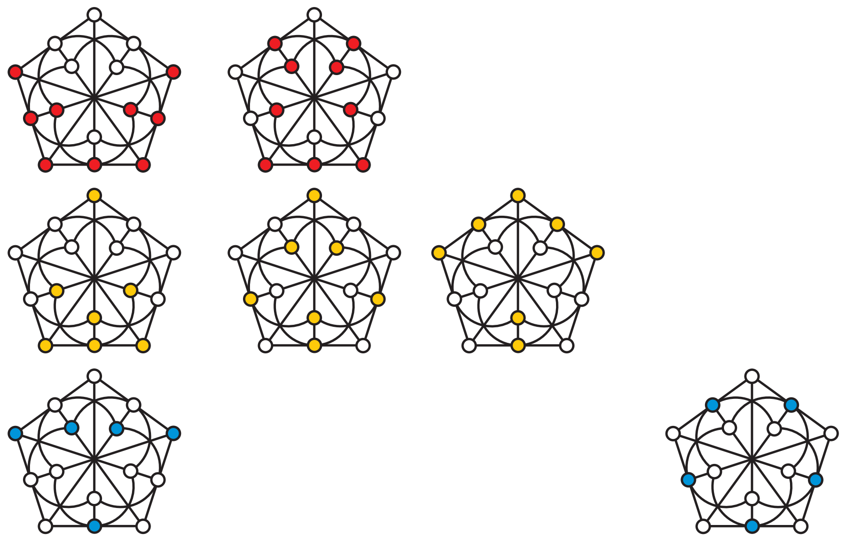


Figure 8.3: The three kinds of geometric hyperplanes of $W(3,2)$

segments as well as by the segments of circles; note that not every intersection of two segments counts for a point of the doily. The upper panel shows grids (red bullets), the middle panel perpsets (yellow bullets) and the bottom panel ovoids (blue bullets). Each picture – except that located in the bottom right-hand corner – stands for five different hyperplanes, the four other being obtained from it by its successive rotations through 72 degrees around the center of the pentagon. The 155 lines of $\mathcal{V}(W(3,2))$ split into five distinct types as summarized in Tab. 8.1 which is an overview of the properties of the five different types of lines of $\mathcal{V}(W(3,2))$ in terms of the core (*i.e.*, the set of points common to all the three hyperplanes forming the line) and the types of geometric hyperplanes featured by a generic line of a given type. The last column gives the total number of lines per each type. This is depicted in Fig. 8.4, an illustrative portrayal of representatives (rows) of the five(numbered consecutively from top to bottom) different types of lines of $\mathcal{V}(W(3,2))$, each being uniquely determined by the properties of its core (black bullets). (Tab. 8.1, as well as Fig. 8.3 and Fig. 8.4, were taken from [SLPP09].)

Type	Core	Perps	Ovoids	Grids	#
I	Two Secant Lines	1	0	2	45
II	Single Line	3	0	0	15
III	Tricentric Triad	3	0	0	20
IV	Unicentric Triad	1	1	1	60
V	Single Point	1	2	0	15

Table 8.1: Properties of the lines types of $\mathcal{V}(W(3,2))$ in terms of the core

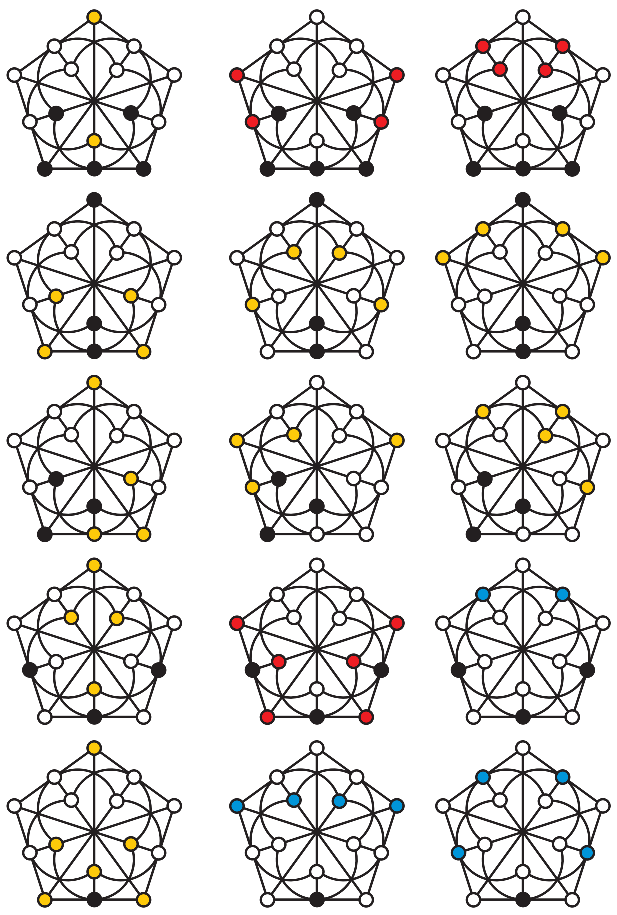


Figure 8.4: Representatives of the lines types of $\mathcal{V}(W(3,2))$

In what follows we will mainly be focused on $W(2N-3,2)$'s that are located in $W(2n-1,2)$. These are, in general, of two different kinds [VL10]. A $W(2N-3,2)$ of the first kind, to be called linear, is isomorphic to the intersection of two perpsets with non-collinear nuclei and their number in $W(2n-1,2)$ is

$$|W|_{W_l} = \frac{1}{3}4^{n-1}(4^n - 1). \tag{8.14}$$

A $W(2n-3,2)$ of the second kind, to be called quadratic, is isomorphic to the intersection of a hyperbolic quadric and an elliptic quadric and $W(2n-1,2)$ features

$$|W|_{W_q} = 4^{n-1}(4^n - 1) \tag{8.15}$$

of them. By way of example, in $W(3,2)$ a linear (resp. quadratic) $W(1,2)$ corresponds to a tricentric (resp. unicentric) triad.

In the sequel, when referring to $W(2n-1,2)$ and its subspaces we will always have in mind the $W(2n-1,2)$ and its subspaces whose points are labelled by n-qubit observables from the set $\mathcal{S}_n$ as expressed by Eqs. (8.11) and (8.12). Moreover, a linear subspace of such $W(2n-1,2)$ will be called *positive* or *negative* according as the (ordinary) product of the observables located in it is $+\mathcal{I}_n$ or $-\mathcal{I}_n$, respectively. Let us illustrate this point taking again the $n = 2$ case. Up to isomorphism, there is just one type of the two-qubit doily. Its six observables of type A are IX, XI, IY, YI, IZ, and ZI and its nine ones of type B are XX, XY, XZ, YX, YY, YZ, ZX, ZY and ZZ, the latter lying on a particular hyperbolic quadric, $\mathcal{Q}^+_{(YY)}(3,2)$. Among the 15 lines only the three lines $\{XX, YY, ZZ\}$, $\{XY, YZ, ZX\}$ and $\{XZ, YX, ZY\}$ are negative, forming also one system of generators of $\mathcal{Q}^+_{(YY)}(3,2)$.

8.3/ $W(3,2)$ AND ITS TWO-QUBIT $W(1,2)$'S

This is a rather trivial case. As already mentioned in Sec. 2, the doily contains three negative lines, of the same $(B-B-B)$ type. Among its $W(1,2)$'s, we find two types of linear ones and three types of quadratic ones, whose properties are summarized in Tab. 8.2 which gives a classification of $W(1,2)$'s living in $W(3,2)$. Column one (T) shows the type, columns two and three (O_A and O_B) indicate the number of observables of corresponding types, and columns four (W_l) and five (W_q) yield, respectively, the number of 'linear' and 'quadratic' $W(1,2)$'s of a given type.

T	O_A	O_B	W_l	W_q
1	0	3	–	6
2	1	2	–	36
3	1	2	18	–
4	2	1	–	18
5	3	0	2	–

Table 8.2: Classification of $W(1,2)$'s living in $W(3,2)$

It is worth noticing that the six quadratic $W(1,2)$'s (i.e., unicentric triads) of Type 1 lie on the distinguished quadric $Q^{+}_{(YY)}(3,2)$, being in fact its six ovoids.

8.4/ $W(5,2)$ AND ITS THREE-QUBIT DOILIES

The space $W(5,2)$ contains 63 points, 315 lines and 135 generators, the latter being all Fano planes. Among the 63 canonical three-qubit observables associated to the points, nine are of type A, 27 are of type B and 27 are of type C. Through an observable of type C there pass six negative lines, all being of type $C-C-B$; the total number of negative lines of this type thus is $\frac{27\times 6}{2} = 81$. Through an observable of type B there pass four negative lines. Of them, three are of the above-mentioned type and the fourth one is of type $B-B-B$; the total number of negative lines of the latter type is $\frac{27\times 1}{3} = 9$. As no negative line features an observable of type A, one finds that the $W(5,2)$ accommodates as many as $(81+9=)$ 90 negative lines.

T	C^{-}	O_A	O_B	O_C	D_l	D_q
1	7	0	7	8	–	81
2	7	0	9	6	27	–
3	6	1	5	9	–	108
4	5	2	5	8	162	–
5	5	2	7	6	–	162
6	4	3	5	7	–	324
7	3	0	9	6	9	–
8	3	0	15	0	–	36
9	3	2	7	6	–	216
10	3	2	9	4	81	–
11	3	4	5	6	54	–
12	3	4	7	4	–	81
13	3	6	9	0	3	–

Table 8.3: Classification of doilies living in $W(5,2)$

When we pass to $W(3,2)$'s, we find a (much) richer structure, because alongside the types of observables we can employ one more parameter, namely the number of negative lines a given $W(3,2)$ contains. In fact, we find that the 336 linear doilies (see Eq. 8.14) fall into six different types and the 1008 quadratic ones (see Eq. 8.15) into seven types; we note in passing that Type 9 splits further into two subtypes depending on whether the two observables of type A do (Type $9A$, 162 members) or do not (Type $9B$, 54 members) commute. This classification is summarized in Tab. 8.3 which gives a classification of doilies living in $W(5,2)$. Column one (T) shows the type, column two (C^{-}) the number of negative lines in a doily of the given type, columns three to fiven(O_A to O_C) indicate the number of observables of corresponding types, and columns six (D_l) and seven (D_q)

yield, respectively, the number of 'linear' and 'quadratic' doilies of a given type. It is also pictorially illustrated in Fig. 8.5, a depiction of representatives – numbered consecutively from left to right, top to bottom – of the 13 different types of three-qubit doilies; Type 1 is top left, Type 13 bottom middle; we also distinguish between Type $9A$ (3rd row, right) and Type $9B$ (4th row, left). The three different types of observables are distinguished by different colors and the negative lines are drawn heavy. It is worth noticing here that there are two different types of doilies (Type 3 and Type 6) exhibiting an even number of negative lines.

The 27 observables of type B lie on an elliptic quadric of $W(5,2)$, which can be defined as follows:

$$Q^{-}_{(YYY)}(5,2) := x_1^2 + x_1x_4 + x_4^2 + x_2^2 + x_2x_5 + x_5^2 + x_3^2 + x_3x_6 + x_6^2 = 0.$$

Here, we took a coordinate basis of $W(5,2)$ in which the symplectic form $\sigma(x,y)$ is given by Eq. 8.1,

$$\sigma(x,y) = (x_1y_4 - x_4y_1) + (x_2y_5 - x_5y_2) + (x_3y_6 - x_6y_3),$$

so that the correspondence between the 63 three-qubit observables (see Eq. 8.10)

$$\mathcal{S}_3 = \{G_1 \otimes G_2 \otimes G_3 : \ G_j \in \{I,X,Y,Z\}, \ j \in \{1,2,3\}\} \backslash \mathcal{I}_3$$

and the 63 points of $W(5,2)$ is of the form (see Eq. 8.11)

$$G_j \leftrightarrow (x_j, x_{j+3}), \ j \in \{1,2,3\},$$

taking also into account Eq. 8.12.

This special quadric $Q^{-}_{(YYY)}(5,2)$, like any non-degenerate quadric, is a *geometric hyperplane* of $W(5,2)$. As a doily is also a *subgeometry* of $W(5,2)$, it either lies fully in $Q^{-}_{(YYY)}(5,2)$ (Type 8), or shares with $Q^{-}_{(YYY)}(5,2)$ a set of points that form a geometric hyperplane; an ovoid (Types 3, 4, 6 and 11), a perpset (Types 1, 5, 9 and 12) and a grid (Types 2, 7, 10 and 13). One also observes that no quadratic doily shares a grid with $Q^{-}_{(YYY)}(5,2)$.

In addition to the distinguished elliptic quadric, there are also three distinguished hyperbolic quadrics in $W(5,2)$, namely: the quadric whose 35 observables feature either two X's or no X,

$$Q^{+}_{(ZZZ)}(5,2) := x_4^2 + x_5^2 + x_6^2 + x_1x_4 + x_2x_5 + x_3x_6 = 0,$$

the one whose 35 observables feature either two Y's or no Y (see Eq. 8.13),

$$Q^{+}_{(III)}(5,2) := x_1x_4 + x_2x_5 + x_3x_6 = 0,$$

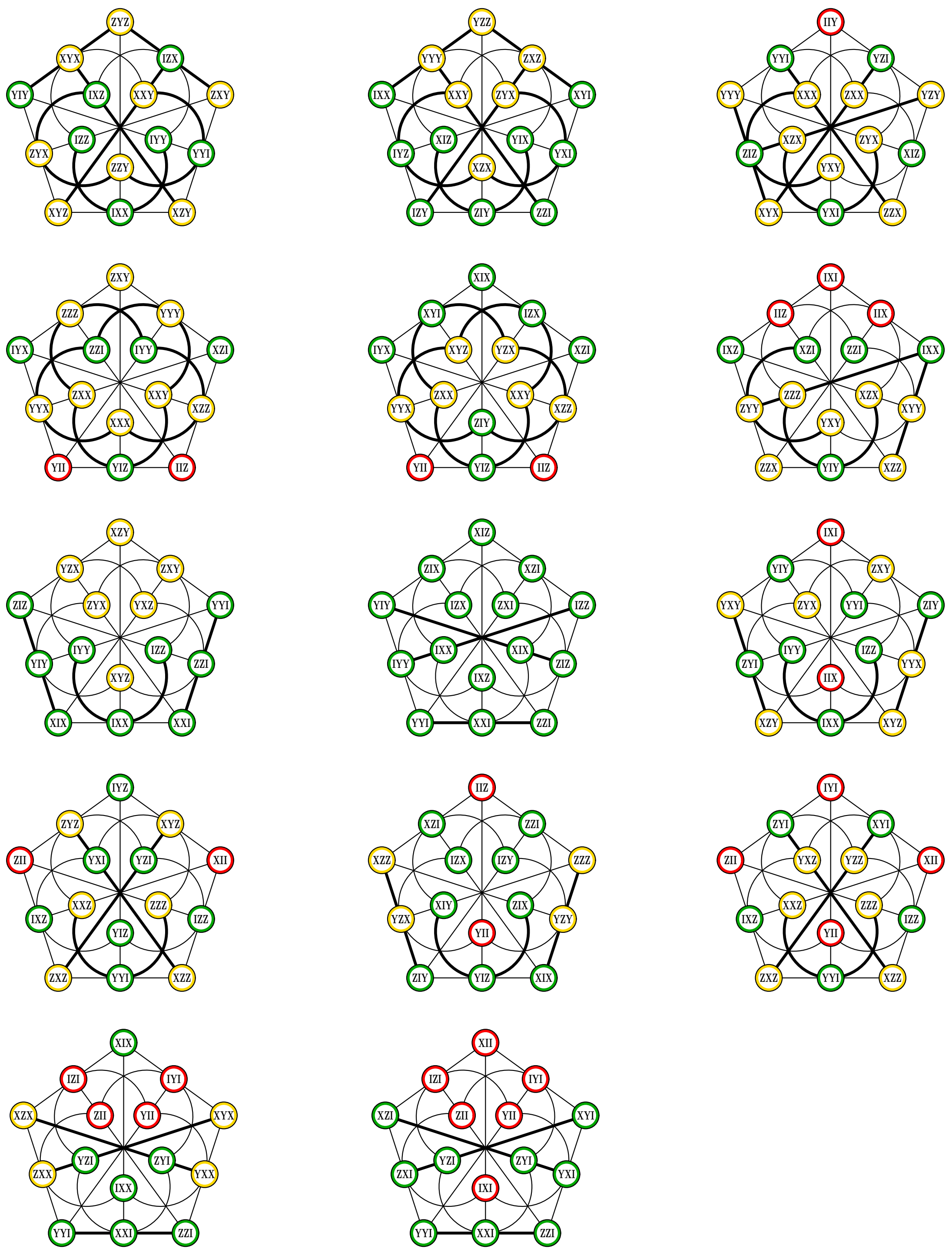


Figure 8.5: Representatives of the types of three-qubit doilies

and the one whose 35 observables feature either two Z's or no Z,

$$Q^{+}_{(XXX)}(5,2) := x_1^2 + x_2^2 + x_3^2 + x_1x_4 + x_2x_5 + x_3x_6 = 0.$$

Accordingly, there are three distinguished doilies of Type 8, namely the ones the quadric

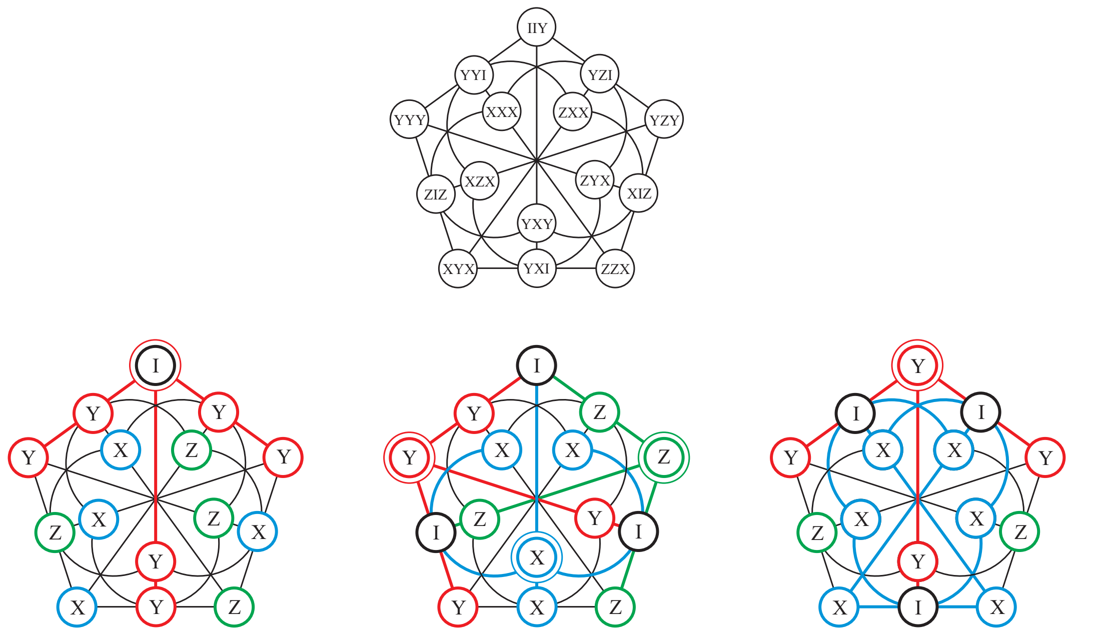


Figure 8.6: Three-qubit doily decomposition into three single-qubit residuals

$Q^{-}_{(YYY)}(5,2)$ shares with these three hyperbolic quadrics.

Take the two-qubit doily. Add formally to each observable, at the same position, the same mark from the set $\{X,Y,Z\}$. Pick up a geometric hyperplane in this three-qubit labeled doily, and replace by I the added mark in each observable that belongs to this geometric hyperplane. One obviously gets a three-qubit doily. Now, there are 31 geometric hyperplanes in the doily, three possibilities (X,Y,Z) to pick up a mark, and three possibilities (left, middle, right) where to insert the mark; so there will be $31\times3\times3=279$ doilies created this way. In particular, out of the $15\times9=135$ doilies 'induced' by perpsets, 81 are of Type 10 and 54 of Type 11; out of the $10\times9=90$ doilies 'generated' by grids, 81 are of Type 12 and 9 of Type 8; finally, the $6\times9=54$ doilies stemming from ovoids are all of the same type $9B$. So, if we look at Tab. 8.3, all doilies of Types 1 to 7, 27 doilies of type 8 and all doilies of type $9A$ can be regarded as 'genuine' three-qubit guys, 9 doilies of type 8 that originate from grids (henceforth referred to as Type $8'$) and all doilies of types $9B$ to 13 can be viewed as 'built from the two-qubit guy'; with Type 13 doilies being even more two-qubit-like.

This stratification of three-qubit doilies can also be spotted in a different way. Take a representative doily of a particular type, for example that of Type 3 depicted in Fig. 8.6, top. From its three-qubit labels, keep first only the left mark (bottom left figure), then the middle mark (bottom middle figure) and, finally, the right mark (bottom right figure). In each of these three 'residual' doilies it is easy to see that if you take the points featuring a given non-trivial mark (i.e., X, Y or Z) together with the points featuring I, these always form a geometric hyperplane, and the whole set form a Veldkamp line of the doily where

T	2cl	le	ttr	utr	pt	ov	ps	gr	fl
1	1	–	–	–	2	–	–	–	–
2	–	3	–	–	–	–	–	–	–
3	–	–	1	1	1	–	–	–	–
4	–	1	2	–	–	–	–	–	–
5	1	–	–	2	–	–	–	–	–
6	1	–	1	1	–	–	–	–	–
7	–	3	–	–	–	–	–	–	–
8	3	–	–	–	–	–	–	–	–
$9A$	1	–	–	2	–	–	–	–	–
$8'$	–	–	2	–	–	–	–	1	–
$9B$	–	–	2	–	–	1	–	–	–
10	–	–	2	–	–	–	1	–	–
11	–	–	2	–	–	–	1	–	–
12	–	–	2	–	–	–	–	1	–
13	–	–	2	–	–	–	–	–	1

Table 8.4: A refined classification of doilies living in $W(5,2)$

the points featuring I represent its core! Fig. 8.6 depicts a formal decomposition of a three-qubit doily (top) into three 'single-qubit residuals' (bottom). In each doily of the bottom row, the three geometric hyperplanes forming a Veldkamp line are distinguished by different colors, with their common points being drawn black; also, the nuclei of perpsets are represented by double circles. Employing Tab. 8.1 we readily see that this Veldkamp line is of type V (the core is a single point) for the left residual doily, type III (the core is a tricentric triad) for the middle doily and of type IV (the core is a unicentric triad) for the right one. To account this way for the 13 types of three-qubit doilies, we also need the concept of a trivial Veldkamp line of the doily, *i.e.* a line consisting of a geometric hyperplane counted twice and the full doily, which exactly accounts for those doilies 'generated' by the two-qubit doily! This classification is summarized in Tab. 8.4 giving a refined classification of doilies living in $W(5,2)$. We use the following abbreviations for the cores of Veldkamp lines: $2cl$ – two concurrent lines, le – line, ttr – tricentric triad, utr – unicentric triad, pt – point, ov – ovoid, ps – perpset, gr – grid and fl stands for the full doily. Here, columns two to six give the number of ordinary Veldkamp lines of a given type, columns seven to nine show the same for trivial Veldkamp lines and the last column corresponds to the degenerate case when all the points of a residual doily bear the label I. Note that all doilies stemming from the two-qubit doily (*i.e.*, Types $8'$ to 13) feature ordinary Veldkamp lines of the same type.

Using our Magma program presented in Chap. 7, we have also found out a very interesting property that given a doily and any geometric hyperplane in it, there are three other doilies having the same geometric hyperplane. Fig. 8.7 serves as a visualization of this fact when the common geometric hyperplane is an ovoid (bold-faced). The top doily is of

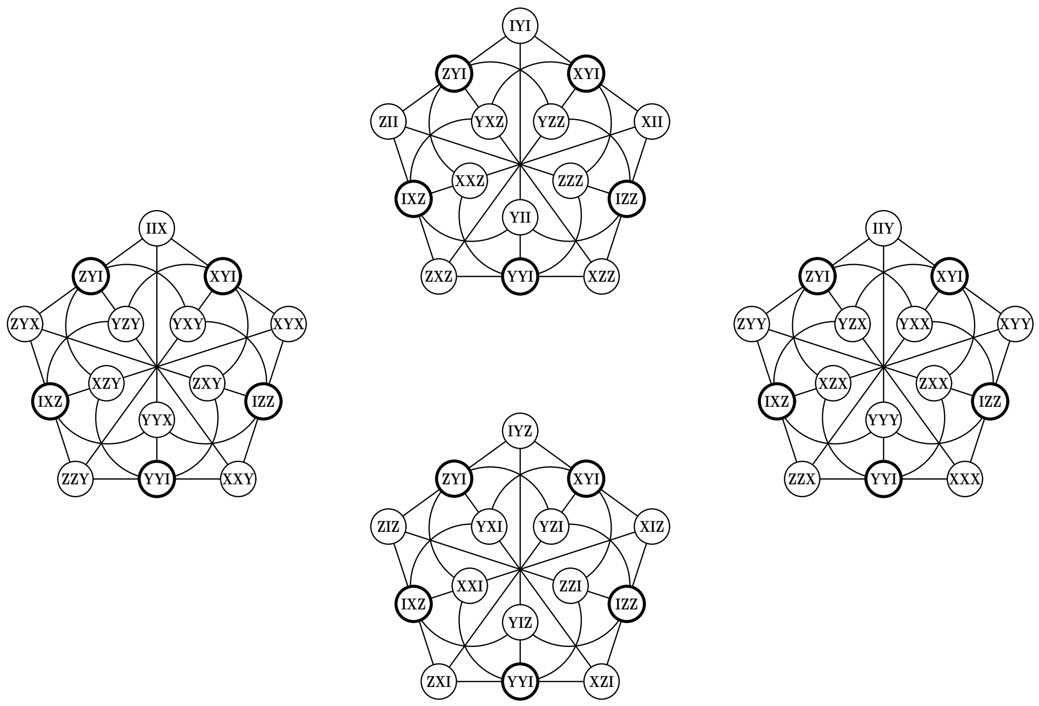


Figure 8.7: Four different doilies sharing an ovoid

Type 11, the bottom one of Type 8, and both the left and right doilies are of Type 3. The four doilies sharing a geometric hyperplane, however, do not stand on the same footing. This is quite easy to spot from our example depicted in Fig. 8.7. A point of the doily is collinear with three distinct points of an ovoid, the three points forming a unicentric triad. Let us pick up such a triad, say $\{ZYI, XYI, YYI\}$ and look for its centers in each of the four doilies. These are IYI (top doily), IIX (left doily), IIY (right doily) and IYZ(bottom one). We see that the last three observables are mutually anticommuting, whereas the first observable commutes with each of them. This property is found to hold for each of $\binom{5}{3} = 10$ triads contained in an ovoid. Hence, the top doily of Fig. 8.7 has indeed a different footing than the remaining three. A similar $3 + 1$ split up is also observed in any quadruple of doilies having a grid in common because a point of the doily is also collinear with three points of a grid that form a unicentric triad. However, when the shared hyperplane is a perpset, one gets a different, namely $2 + 2$ split, because in this case the corresponding triple of points forms a tricentric triad.

Among the 13 different types of three-qubit doilies, there is one type, namely Type 3, which has two remarkable properties. The first property is that there is one point (to be called a deep point) such that all three lines passing through it are negative. Let's take a representative doily of such a type shown in Fig. 8.5, 1st row right. The deep point is ZIZ. Then one sees that there are just two points (to be called zero-points) such that neither of them lies on a negative line; one is IIY and the other is XIZ. These two points and the deep point form in the doily a tricentric triad, hence a copy of 'linear' $W(1, 2)$! The second property is related to the fact that through each observable of type B there pass four negative lines. Three of them are such that each features one observable of type B and two observables of type C, whereas the remaining one consists of all observables of type B. Written vertically, the four negative lines passing through our deep point ZIZ are:

$$\left|\begin{array}{ccc|c} ZIZ & ZIZ & ZIZ & ZIZ \\ XXX & XYX & XZX & XIX \\ YXY & YYY & YZY & YIY \end{array}\right|$$

We see that the three lines that are located in the doily are of the same type, viz. $B-C-C$. If we include also the fourth negative line, viz. the $B-B-B$ one, we get what we can call a ‘doily with a tail.’ Taking into account the above-mentioned four-doilies-per-hyperplane property, we see that there are altogether 12 doilies, four per each observable, having the same tail and all being of Type 3.

8.5/ $W(7,2)$ AND ITS FOUR-QUBIT $W(5,2)$

The space $W(7,2)$ possesses 255 points, 5355 lines, 11475 planes and 2295 generators, the latter being all $PG(3,2)$'s. Among the 255 canonical four-qubit observables associated to the points, 12 are of type A, 54 of type B, 108 of type C and 81 of type D. Through an observable of type D there pass: four negative lines of type $D-D-D$, totaling to $\frac{81\times4}{3}=108$; 12 negative lines of type $D-D-B$, totaling to $\frac{81\times12}{2}=486$; and 12 negative lines of type $D-C-C$, totaling to $81\times12=972$. Through an observable of type C there pass, apart from the above-mentioned lines of type $D-C-C$, six negative lines of type $C-C-B$, totaling to $\frac{108\times6}{2}=324$. Through an observable of type B there passes, apart from the already discussed two types of lines, a single negative line of type $B-B-B$, the total number of such lines being $\frac{54\times1}{3}=18$. Since no negative line can contain an observable of type A, the four-qubit $W(7,2)$ thus exhibits five distinct types of negative lines whose total number is $(108+486+972+324+18=)$ 1908.

When it comes to $W(5,2)$'s, we find 11 types among their 5440 linear members and as many as 18 types among their 16320 quadratic cousins – as summarized in Tab. 8.5 which gives a classification of $W(5,2)$'s living in $W(7,2)$. Column one (T) shows the type, column two (C^-) the number of negative lines in a $W(5,2)$ of the given type, columns three to six (O_A to O_D) indicate the number of observables featuring three I's, two I's, one I or no I, respectively, columns seven (W_l) and eight (W_q) yield, respectively, the number of ‘linear’ and ‘quadratic’ $W(5,2)$'s of a given type, the last but one column depicts the type of intersection of a representative $W(5,2)$ with the distinguished hyperbolic quadric and the last column indicates the type of geometric hyperplane featuring the trivial mark (I) for composite $W(5,2)$'s. It represents no difficulty to check that 54 observables of type B

T	C^-	O_A	O_B	O_C	O_D	W_l	W_q	Int	GH
1	130	3	9	33	18	108	–	ell	– – –
2	126	0	24	0	39	–	108	full	– – –
3	126	1	13	27	22	–	1944	hyp	– – –
4	126	2	10	30	21	–	1620	perp	– – –
5	122	1	15	27	20	972	–	hyp	– – –
6	122	2	10	30	21	–	648	perp	– – –
7	118	0	16	32	15	–	324	perp	– – –
8	118	3	9	33	18	648	–	ell	– – –
9	118	3	11	25	24	–	1296	hyp	– – –
10	114	1	15	27	20	324	–	hyp	– – –
11	114	1	17	27	18	–	216	hyp	– – –
12	114	3	13	25	22	1944	–	hyp	– – –
13	114	4	12	28	19	–	1944	perp	– – –
14	110	3	15	25	20	–	1944	hyp	– – –
15	110	5	11	23	24	648	–	hyp	– – –
16	106	5	13	23	22	–	1944	hyp	– – –
17	102	1	21	27	14	–	648	hyp	– – –
18	102	2	18	30	13	–	324	perp	– – –
19	102	3	15	25	20	–	648	hyp	– – –
20	102	4	12	28	19	–	1944	perp	– – –
21	90	0	36	0	27	–	12	full	ell: $O = YYY$
22	90	2	22	30	9	–	108	perp	hyp: all 9 O's featuring two Y's
23	90	3	9	33	18	36	–	ell	– – –
24	90	3	21	25	14	324	–	hyp	perp: all 27 O's of type C
25	90	4	16	28	15	–	324	perp	ell: all 27 O's featuring one Y
26	90	5	15	31	12	324	–	ell	perp: all 27 O's of type B
27	90	6	18	26	13	–	324	perp	hyp: 26 O's having no Y + III
28	90	7	17	21	18	108	–	hyp	perp: all 9 O's of type A
29	90	9	27	27	0	4	–	ell	full $W(5,2)$

Table 8.5: Classification of $W(5,2)$'s living in $W(7,2)$

and 81 ones of type D lie on a particular hyperbolic quadric in $W(7,2)$, to be referred to as the distinguished hyperbolic quadric $Q^+_{(YYYY)}(7,2)$, which is also a geometric hyperplane in the latter space. A $W(5,2)$ either lies fully in this quadric (Types 2 and 21) or shares with it a set of points that forms a geometric hyperplane. Hence, the sum of O_B and O_D in each row of Tab. 8.5 must be one of the following numbers: 27 (when the hyperplane of $W(5,2)$ is an elliptic quadric), 31 (a perpset) and/or 35 (a hyperbolic quadric); for the reader's convenience, the type of such geometric hyperplane is explicitly listed in column 9 of Tab. 8.5. One sees that no linear $W(5,2)$ shares with $Q^+_{(YYYY)}(7,2)$ a perpset and no quadratic $W(5,2)$ cuts this distinguished quadric in an elliptic quadric. Comparing Tab. 8.5 with Tab. 8.3 one readily discerns that whereas $W(3,2)$'s in $W(5,2)$ are endowed with both an even and odd number of negative lines, for $W(5,2)$'s in $W(7,2)$ this number is always even; in addition, the difference in C^- for any two distinct types of four-qubit $W(5,2)$'s is a multiple of four.

Let us have a closer look at $W(5,2)$'s featuring 90 (i.e., the smallest possible number of) negative lines. We can easily show that almost all of them originate from the three-qubit $W(5,2)$. First, by adding I to each three-qubit observable at the same position we get the four trivial four-qubit $W(5,2)$'s of Type 29. Next, adding to each observable at the same position a mark from the set $\{X,Y,Z\}$, picking up a geometric hyperplane in this four-qubit labeled $W(5,2)$ and replacing by I the added mark of each observable in the geometric hyperplane one gets a four-qubit $W(5,2)$ with 90 negative lines. Now, there are 28 (# of elliptic quadrics) + 36 (# of hyperbolic quadrics) + 63 (# of perpsets) = 127 geometric hyperplanes in the $W(5,2)$, three possibilities (X,Y,Z) to pick up a mark, and four possibilities (left, middle-left, middle-right, right) where to insert the mark. So, there will be $127 \times 3 \times 4 = 1524$ four-qubit $W(5,2)$'s created this way, which only falls short by 36 the total number of $W(5,2)$'s endowed with 90 negative lines (the four guys of Type 29 being, of course, disregarded). A concise summary is given in the last column of Tab. 8.5, where the type of geometric hyperplane is further specified by the character/type of the associated (three-qubit) observable. One observes that Type 23 is the only irreducible type of $W(5,2)$'s having 90 negative lines.

We shall illustrate this process by a couple of examples. Let us start with the perpset of the three-qubit $W(5,2)$ whose nucleus is an observable of type A, say XII. Out of 31 observables commuting with this observable there are 7 of type A (XII, IXI, IIX, IYI, IIY, IZI and IIZ), 15 of type B (IXX, IXY, IXZ, XXI, XIX, IYX, IYY, IYZ, XYI, XIY, IZX, IZY, IZZ, XZI, and XIZ) and 9 of type C (XXX, XXY, XXZ, XYX, XYY, XYZ, XZX, XZY, and XZZ). Hence, out of 32 observables off the perp, there will be $9-7=2$ of type A, $27-15=12$ of type B and $27-9=18$ of type C:

$\widehat{Q}_{(XII)}$	O_A	O_B	O_C
on	7	15	9
off	2	12	18

.

Next, each observable of the perpset acquires a trivial mark I and hence goes into the four-qubit observable of the same type. However, an observable lying off the perpset gets a non-trivial label X, Y or Z and so yields the four-qubit observable of the subsequent type; that is, $O_A^{(3)} \to O_B^{(4)}$, $O_B^{(3)} \to O_C^{(4)}$ and $O_C^{(3)} \to O_D^{(4)}$. Hence, in our case we get:

$(\widehat{Q}_{(XII)})$	O_A	O_B	O_C	O_D
(on – type intact)	7	15	9	0
(off – type shifted)	0	2	12	18
Total	$\textcircled{7}$	$\textcircled{17}$	$\textcircled{21}$	$\textcircled{18}$

.

Comparing with Tab. 8.5 we see that this is a four-qubit $W(5,2)$ of Type 28.

As the second example we shall take the case when the geometric hyperplane of $W(5,2)$ is an elliptic quadric generated by an antisymmetric observable of type B, say YXI. This quadric, $Q^{-}_{(YXI)}(5,2)$, consists of all symmetric observables that commute with YXI and all antisymmetric observables that anticommute with YXI. In particular, it contains 4 observables of type A (IXI, IIX, IIZ and IYI), 11 observables of type B (XZI, ZZI, YIY, IXX, IXZ, YZI, IYX, IYZ, XIY, ZIY and IZY) and 12 observables of type C (XZX, ZZX, XZZ, ZZZ, YXY, XYY, ZYY, YZX, YZZ, XXY, ZXY and YYY). So, out of 36 observables off the quadric, there will be 5, 16 and 15 of type A, B and C, respectively. In a succinct form,

$Q^{-}_{(YXI)}(5,2)$	O_A	O_B	O_C
on	4	11	12
off	5	16	15

.

From this it follows that the corresponding four-qubit $W(5,2)$ is of Type 25:

$(Q^{-}_{(YXI)}(5,2))$	O_A	O_B	O_C	O_D
(on – type intact)	4	11	12	0
(off – type shifted)	0	5	16	15
Total	④	⑯	㉘	⑮

.

8.6/ Conclusion

We have introduced a remarkable observable-based taxonomy of subspaces of $W(2n-1,2)$, $2 \leq n \leq 4$, whose rank is just one less than that of the ambient space. Alongside the distribution of various types of observables, an important parameter of the classification was the number of negative lines contained in a subspace. As already mentioned in the introduction, this latter parameter is essential in checking whether a given finite geometric configuration is contextual or not. For example, our preliminary analysis shows that all three-qubit and four-qubit doilies are, like their two-qubit sibling, contextual. As a future work we plan to address this question in more detail, employing also the degree of contextuality, for a variety of other symplectic subspaces. However, when approaching this way subspaces of higher rank, it would be natural to include as parameters the number of negative linear subspaces of every viable dimension from 1 to $n-2$, *i.e.* consider negative lines, negative planes, ..., negative generators; so, already in the case of $n = 4$ we can add one more parameter, the number of negative planes a four-qubit $W(5,2)$ is endowed with, to get an interesting refinement of our Tab. 8.5. As the three-qubit $W(5,2)$ features 54 negative planes [SHJ20], each composite four-qubit $W(5,2)$ must have the same number of negative planes; in connection with this fact it would be interesting to check whether also each irreducible four-qubit $W(5,2)$ having 90 lines (Type 23) enjoys this property.

Another interesting extension/variation of our taxonomy would be to take into account the number of negative lines passing through a point of the subspace. Let us call this number the order of a point and for each subspace $W(2s-1,2)$ define the following string of parameters $[p_0, p_1, p_2, \ldots, p_{2^s-1}]$, where p_k, $0 \leq k \leq 2^s - 1$, stands for the number of points of order k the subspace contains. Applying this to three-qubit doilies ($s = 2$), we find the following five patterns (as readily discerned from Fig. 8.3): $[0, 9, 6, 0]$ (Types 1 and 2), $[2, 9, 3, 1]$ (Type 3), $[5, 5, 5, 0]$ (Types 4 and 5), $[6, 6, 3, 0]$ (Type 6) and $[6, 9, 0, 0]$ (Types 7 to 13).

A slightly different possibility of employing our strategy is to analyse other distinguished subgeometries of $W(2n-1,2)$ like, for example, the split Cayley hexagon of order two [PSVM01]. This generalized polygon can be embedded into $W(5,2)$, and in two different ways at that [Coo10], called classical and skew. We have already discerned two distinct kinds of the former, and as many as 13 different types of the latter. Yet, a full understanding of the case requires a more rigorous computer-assisted approach and will, therefore, be treated in a future work.

CONCLUSION

General conclusion

Program verification is a key element for assessing that programs behave in accordance with some formalized requirements. So far, quantum program verification is at its very early stages. In this thesis, I have explored ways of specifying and evaluating two quantum properties, entanglement and contextuality, that prepare their formalization in a quantum program verification tool.

At first, a quantum circuit simulator has been developed in SageMath, allowing us to evaluate quantum state properties at any intermediate execution step, in a similar way to verification method known as runtime assertion checking. This prototype also offers the nice feature of optional choice between floating-point and exact computation, for either faster or more accurate results. With this simulator I presented two example algorithms – Deutsch's algorithm and Deutsch-Jozsa's algorithm (Chap. 3). Deutsch's algorithm was validated using exhaustive testing – a method where every possible input is tested – and Deutsch-Jozsa's algorithm was validated up to $n = 5$ qubits using bounded exhaustive testing. Both of these methods are elementary ways to validate software, only available in some simple cases. This same simulator was used in order to study the variation of entanglement in Grover's algorithm and the QFT using Mermin's polynomials (Chap. 4, [dJH+20, dJH+21]). In this work, we concluded that Mermin's polynomials are a suitable tool to dynamically measure entanglement at each step of a quantum algorithm. Furthermore, entanglement has an interesting behavior in Grover's algorithm: it increases up to a middle point, and then decreases until the end of the algorithm. This property could be used to check the validity of an implementation of Grover's algorithm for example. The behavior of the QFT in that regard is less regular, but we draw a comparison between the measure of entanglement using Mermin's polynomials and using the Cayley hyperdeterminant, showing some similarities, but also some differences. This work was then transposed with Grâce Amouzou to IBM's Qiskit library (Chap. 5) in order to assess whether these results could be transposed to NISQ (Noisy Intermediate-Scale Quantum) processors, resulting in an article on the subject by G. Amouzou, J. Boffelli, H. Jaffali, K. Atchonouglo and F. Holweck [ABJ+20]. At this time, the noise in IBM's quantum processors is still too important for obtaining results corresponding to the simulated ones, but the execution on their simulator yielded the same results as ours. The study of the QFT validated that some gates would affect entanglement, and others wouldn't. This gate property could be used in specification and validation.

Motivated by previous works by two of my thesis supervisors, we then tackled contextuality from the point of view of finite geometry. Jessy Colonval and I introduced ourselves to the field by implementing some known geometries generation algorithms in Magma (Chap. 6, [Cd19]). We noticed that the mathematical community still lacked some important coding practices, resulting in imprecise results that we outlined. This work was a warm-up in finite geometries generation with Magma, followed up by the generation of finite geometries said to be *quantum* for their link with Mermin-Peres geometries. These geometries were thoroughly studied in Chap. 7 and 8. We first worked on families of geometries, assessing if their members were contextual geometries or not (Chap. 7, [dHGM21a, dHGM21b]). We found out that, for $n \in [2..5]$ qubits, the lines of $W(2n-1,2)$ – the symplectic space of n qubits, a space encoding the commutation relations in the n-qubits Pauli group–, their restrictions to hyperbolic quadrics and their restrictions to elliptic quadrics are all contextual, but $W(2n-1,2)$'s generators and $W(2n-1,2)$'s lines restricted to the perpsets are never contextual. Then, we studied the structure of $W(2n-1,2)$, using a newly defined signature for the geometries, being made up of the number of negative lines and the number of observables with i identities with $i \in [0..n-1]$ for each geometry (Chap. 8, [SdHG21]). This study is complemented by correspondences between $W(2n-1,2)$, $W(2(n-1)-1,2)$ and $W(2(n-2)-1,2)$ increasing geometries of a given number of qubits with strategically placed operators to obtain geometries of the size above, and is specifically performed for geometries using 2 to 5 qubits observables. These studies aim at better understanding contextuality in order to find remarkable properties that could be used for specifying contextuality in quantum programs, but they are only a first step on a long road.

Perspectives

Many of the tackled subjects are begging for a follow-up. In particular, Chap. 4 – evaluation of entanglement and non-locality in Grover's algorithm and the QFT with Mermin's polynomials – led us to a work in progress on formalizing Mermin's polynomials and the QFT in the proof assistant Coq[1]. An example of the tasks from this work is given in Appendix D where I show some Coq code proving that Mermin's polynomials are Hermitian operators. Once finished, this work could be extended by a formal proof of the growing-then-falling aspect of entanglement during Grover's algorithm. The QFT seems harder to study on that regard because of its irregular behavior, but Grover's algorithm is a good candidate. Furthermore, since our study, Grover's algorithm was implemented and its correctness as of finding the searched element was proven in SQIRE (a Coq library for quantum circuits formalization), which means we could heavily rely on this work, and focus on Mermin's polynomials implementation. This could also be tackled on another formal verification software, such as Why3 where automation may help speed up the process. In particular, a recent Why3 library called Qbricks [CBB+21] would fit in very well in this project.

We also started to formalize quantum geometries in Why3, with the aim of proving general

[1] https://coq.inria.fr/

properties for all numbers of qubits. This work is also in progress, and should be finished. Additionally, the space containing the geometries has a very rich structure, that could be further explored. We keep in mind that, even though we studied quantum contextuality through these geometries, contextuality can also be highlighted in a more general way by protocols, like those used in quantum games. This means that programs could be studied under this aspect, allowing us eventually to specify them using contextuality. In this case, studying the link between our approach of contextuality and protocols involving contextuality may allow us to use our work in quantum geometries for quantum programs specification.

Furthermore, our work gives rise to several conjectures, that we validated only for small numbers n of qubits. This could be proven for all n, but if close enough of a formal representation of the problem, this is also a place where formal proof softwares would shine. An example of such conjecture is the fact that all $W(2n-1,2)$'s lines and its quadrics form contextual geometries for all numbers n of qubits.

IV

Appendices

A

Deutsch-Jozsa oracle

This chapter gives a formula to quickly compute the matrix corresponding to the oracle used in the Deutsch-Jozsa algorithm. This formula is followed by its proof, detailed enough to be formalized in a proof language later on if need be.

A.1/ Formula

First of all, let's introduce some notations :

- $\mathbb{B} = \{0, 1\}$,
- $n \in \mathbb{N}^*$ is the number of bits for the input of the function
- $I = [0..2^n - 1]$ and $I' = [0..2^{n+1} - 1]$ are intervals that will be regularly used

$f : I \to \mathbb{B}$ is the function for which we want to know if it is constant or balanced.

Definition 3: Deutsch-Jozsa oracle

The oracle $U_f = \left(u_{i,j}\right)_{(i,j)\in I'^2} \in \mathcal{M}_{2^{n+1}}(\mathbb{C})$ is defined as such : $\forall(x, y) \in I \times \mathbb{B}, U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle$.

With this definition, we can now introduce $(C_i)_{i\in I} \in \mathcal{M}_2(\mathbb{C})^n$, where the C_i are defined as such:

$$\forall i \in I, C_i = \begin{pmatrix} u_{2i,2i} & u_{2i,2i+1} \\ u_{2i+1,2i} & u_{2i+1,2i+1} \end{pmatrix}.$$

And we'll use $\{C_i\}$ as the set of the coefficients of C_i.

Let's recall that $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$.

Theorem 1 (*Deutsch-Jozsa oracle formula*): *We can compute U_f with the following formula*

$$\forall x \in I, \begin{cases} f(x) = 0 \implies C_i = I \\ f(x) = 1 \implies C_i = X \\ \forall (i, j) \in I'^2, \neg(\exists k, u_{i,j} \in \{C_k\}) \implies u_{i,j} = 0 \end{cases}$$

Visually, $U_f = \begin{pmatrix} C_0 & & 0 \\ & \ddots & \\ 0 & & C_{2^n-1} \end{pmatrix}$,
and we can summarize the formula as such: U_f is a null matrix except on the diagonal blocks where it can be either I or X depending if $f(x)$ is equal to 0 or 1 respectively.

Yet another way to write this is to see that $U_f = \sum_{i\in I} e_i \otimes C_i$ with $e_i = |i\rangle\langle i|$.

A.2/ Proof

$\forall(x,y) \in I \times \mathbb{B}, |x\rangle \otimes |y\rangle$ is the vector with 1 being its single entry at position $2x+y$. This implies that, given that $U_f\,|x\rangle \otimes |y\rangle = |x\rangle \otimes |y \oplus f(x)\rangle$, $(u_{i,2x+y})_{i\in I'}$ is such a vector where the 1 is in position $2x + (y \oplus f(x))$.

We thus have $u_{2x+(y\oplus f(x)),2x+y} = 1$ and $\forall i \in I - \{2x + (y \oplus f(x))\}, u_{i,2x+y} = 0$

From this, we deduce that there is only one '1' per column: on the $2x+y^{th}$ column, it is at the $2x + (y \oplus f(x))^{th}$ line. Which yields the following result:

$$f(x) = 0 \implies \begin{cases} U_f(|x\rangle \otimes |0\rangle) = |x\rangle \otimes |0\rangle \\ U_f(|x\rangle \otimes |1\rangle) = |x\rangle \otimes |1\rangle \end{cases} \implies \begin{cases} u_{2x,2x} = 1 \\ u_{2x+1,2x+1} = 1 \end{cases} \implies C_x = I$$

and

$$f(x) = 1 \implies \begin{cases} U_f(|x\rangle \otimes |0\rangle) = |x\rangle \otimes |1\rangle) \\ U_f(|x\rangle \otimes |1\rangle) = |x\rangle \otimes |0\rangle) \end{cases} \implies \begin{cases} u_{2x+1,2x} = 1 \\ u_{2x,2x+1} = 1 \end{cases} \implies C_x = X$$

A.3/ Proof 2

A more calculation oriented proof is given below, using the Kronecker operator ($\delta_i^j = 1$ if $i = j$ and $\delta_i^j = 0$ otherwise).

As seen previously $\forall(x,y) \in I \times \mathbb{B}, |x,y\rangle = \sum_{k\in I'} \delta_k^{2x+y}\,|k\rangle$. Given that $U_f = \sum_{(i,j)\in I'^2} u_{i,j}\,|i\rangle\langle j|$, we get:

$$\begin{aligned} U_f(|x\rangle \otimes |y\rangle) &= \sum_{i,j,k \in I'} u_{i,j} \delta_k^{2x+y} |i\rangle \langle j|k\rangle \\ &= \sum_{i,j \in I'} u_{i,j} \delta_j^{2x+y} |i\rangle \\ &= \sum_{i \in I'} u_{i,2x+y} |i\rangle \end{aligned}$$

Given that $|x\rangle \otimes |y \oplus f(x)\rangle) = \sum_{k \in I'} \delta_k^{2x+(y \oplus f(x))} |k\rangle$ and $U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle)$, we have $\forall (i, x) \in I' \times I, u_{i,2x+y} = \delta_i^{2x+(y \oplus f(x))}$. Which results in

$$\begin{cases} f(x) = 0 \implies \forall k \in \{2x, 2x+1\}, u_{k,2x+y} = \delta_k^{2x+y} \\ f(x) = 1 \implies \forall k \in \{2x, 2x+1\}, u_{k,2x+y} = \delta_k^{2x+(y \oplus 1)} \end{cases}$$

Hence the result.

This second proof was added because it may be easier to formalize.

B

Explicit states for Grover's algorithm

Proposition 4: *[HJN16, Observation 1] The state $|\varphi_k\rangle$ after k iterations of Grover's algorithm can be written as follows:*

$$|\varphi_k\rangle = \tilde{\alpha}_k \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \tilde{\beta}_k |+\rangle^{\otimes n} \tag{B.1}$$

with $\tilde{\alpha}_k = \frac{\cos\left(\frac{2k+1}{2}\theta\right)}{\sqrt{|S|}} - \frac{\sin\left(\frac{2k+1}{2}\theta\right)}{\sqrt{N-|S|}}$ *and* $\tilde{\beta}_k = 2^{n/2}\frac{\sin\left(\frac{2k+1}{2}\theta\right)}{\sqrt{N-|S|}}$.

Proof. With $|\varphi_0\rangle = |+\rangle^{\otimes n}$, we can write:

$$|\varphi_k\rangle = \mathcal{L}^k |\varphi_0\rangle = \frac{a_k}{\sqrt{|S|}} \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \frac{b_k}{\sqrt{N-|S|}} \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle$$

where $\mathcal{L}$ is the loop (oracle and diffusion operator) in Grover's algorithm.

The oracle is a reflection about $(\sum_{\mathbf{x}\in S} |\mathbf{x}\rangle)^{\perp} = \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle$ and the diffusion operator is a reflection about $|+\rangle^{\otimes n}$. The composition of these two symmetries is a rotation whose angle θ is the double of the angle between $\sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle$ and $|+\rangle^{\otimes n}$. So,

$$\begin{array}{rcl}
|+\rangle^{\otimes n} & = & \frac{1}{\sqrt{|S|}} \sin\left(\frac{\theta}{2}\right) \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \frac{1}{\sqrt{N-|S|}} \cos\left(\frac{\theta}{2}\right) \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle \\
\frac{1}{\sqrt{N}} \left(\sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle\right) & = & \frac{1}{\sqrt{|S|}} \sin\left(\frac{\theta}{2}\right) \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \frac{1}{\sqrt{N-|S|}} \cos\left(\frac{\theta}{2}\right) \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle \\
\frac{1}{\sqrt{N}} \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle & = & \frac{1}{\sqrt{|S|}} \sin\left(\frac{\theta}{2}\right) \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle \\
\frac{1}{\sqrt{N}} & = & \frac{1}{\sqrt{|S|}} \sin\left(\frac{\theta}{2}\right) \\
\sin\left(\frac{\theta}{2}\right) & = & \sqrt{\frac{|S|}{N}}.
\end{array}$$

The fact that $\mathcal{L}$ is a rotation of angle θ gives $a_k = \sin(\theta_k)$ and $b_k = \cos(\theta_k)$ with $\theta_k = k\theta + \theta/2$. Equation (4.1) then comes from $\alpha_k = \frac{1}{\sqrt{|S|}} \sin\left(\frac{2k+1}{2}\theta\right)$ and $\beta_k = \frac{1}{\sqrt{N-|S|}} \cos\left(\frac{2k+1}{2}\theta\right)$.

With this, we can now take $\tilde{\alpha}_k = \alpha_k - \beta_k$ and $\tilde{\beta}_k = 2^{n/2}\beta_k$ which gives us

$$\begin{array}{rcl} |\varphi_k\rangle & = & \alpha_k \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \beta_k \sum_{\mathbf{x}\notin S} |\mathbf{x}\rangle \\ & = & (\alpha_k - \beta_k) \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \beta_k \sum_{\mathbf{x}=0}^{N-1} |\mathbf{x}\rangle \\ & = & \tilde{\alpha}_k \sum_{\mathbf{x}\in S} |\mathbf{x}\rangle + \tilde{\beta}_k |+\rangle^{\otimes n} \end{array}$$

since $|+\rangle^{\otimes n} = \left(\frac{1}{\sqrt{2}}\right)^n \sum_{\mathbf{x}=0}^{N-1} |\mathbf{x}\rangle$.

Proposition 5: *In Proposition 4, $\tilde{\alpha}_k$ increases for k between 0 and $\frac{\pi}{4}\sqrt{\frac{N}{|S|}} - \frac{1}{2}$ and $\tilde{\beta}_k$ decreases on the same interval.*

Proof. The optimal number of iterations of the loop $\mathcal{L}$ in Grover's algorithm is the smallest value k_{opt} of k such that $a_k = 1$, *i.e.*, $\theta_{k_{opt}} = \pi/2$. With $|S| \ll N$, $\sin(\theta/2) = \sqrt{|S|/N}$ gives $\theta \approx 2\sqrt{|S|/N}$ and $\theta_k \approx (2k+1)\sqrt{|S|/N}$. Finally $(2k_{opt}+1)\sqrt{|S|/N}$ optimally approximates $\pi/2$ if $k_{opt} = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{|S|}} - \frac{1}{2} \right\rceil = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{|S|}} \right\rfloor$.

Moreover, $a_k = \sin(\theta_k)$ and $\alpha_k = \frac{1}{\sqrt{|S|}} a_k$ are increasing and $b_k = \cos(\theta_k)$ and $\beta_k = \frac{1}{\sqrt{N-|S|}} b_k$ are decreasing for k from 0 to $\left(\frac{\pi}{4}\sqrt{\frac{N}{|S|}} - \frac{1}{2}\right)$. From the expressions $\tilde{\alpha}_k = \alpha_k - \beta_k$ and $\tilde{\beta}_k = 2^{n/2}\beta_k$, we get the result of the proposition.

C

CAYLEY HYPERDETERMINANT Δ_{2222}

Let $|\varphi\rangle = \sum_{i,j,k,l\in\{0,1\}} a_{i,j,k,l}\,|ijkl\rangle$ be a four-qubit state. The algebra of polynomial invariants for the four-qubit Hilbert space can be generated by the four polynomials H, L, M and D defined as follows [LT03]:

$$H = a_{0000}a_{1111} - a_{1000}a_{0111} - a_{0100}a_{1011} + a_{1100}a_{0011}$$
$$-a_{0010}a_{1101} + a_{1010}a_{0101} + a_{0110}a_{1001} - a_{1110}a_{0001}$$

is an invariant of degree 2.

$$L = \begin{vmatrix} a_{0000} & a_{0010} & a_{0001} & a_{0011} \\ a_{1000} & a_{1010} & a_{1001} & a_{1011} \\ a_{0100} & a_{0110} & a_{0101} & a_{0111} \\ a_{1100} & a_{1110} & a_{1101} & a_{1111} \end{vmatrix} \quad \text{and} \quad M = \begin{vmatrix} a_{0000} & a_{0001} & a_{0100} & a_{0101} \\ a_{1000} & a_{1001} & a_{1100} & a_{1101} \\ a_{0010} & a_{0011} & a_{0110} & a_{0111} \\ a_{1010} & a_{1011} & a_{1110} & a_{1111} \end{vmatrix}$$

are two invariants of degree 4.

Consider the partial derivative

$$b_{xt} := \det\left(\frac{\partial^2 A}{\partial y_i \partial z_j}\right)$$

of the quadrilinear form $A = \sum_{i,j,k,l\in\{0,1\}} a_{i,j,k,l}x_i y_j z_k t_l$ with respect to the variables y and z. This quadratic form with variables x and t can be interpreted as a bilinear form on the three-dimensional space $\mathrm{Sym}^2(\mathbb{C}^2)$, *i.e.*, there is a 3×3 matrix B_{xt} satisfying

$$b_{xt} = [x_0^2, x_0x_1, x_1^2]\; B_{xt} \begin{bmatrix} t_0^2 \\ t_0t_1 \\ t_1^2 \end{bmatrix}.$$

Then $D = \det(B_{xt})$ is an invariant of degree 6.

Let's introduce the invariant polynomials

$$U = H^2 - 4(L - M), \qquad V = 12(HD - 2LM),$$

$$S = \frac{1}{12}(U^2 - 2V) \qquad \text{and} \qquad T = \frac{1}{216}(U^3 - 3UV + 216D^2).$$

Then the Cayley hyperdeterminant is [LT03]:

$$\Delta_{2222} = S^3 - 27T^2.$$

D

Coq proofs

This chapter presents two Coq proofs the first one (listing D.1) showing that Mermin polynomials are well formed operators in the sense of SQIRE, and the second (listing D.2) that they are Hermitian. These proofs are not here to be understood entirely, but rather to show what efforts must be made to prove “simple” properties in Coq+SQIRE.

In SQIRE, an operator is a function taking two integers as input, are returning a complex number. This is a way to represent matrices using functions. an operator also has bounds: we are only working in finite spaces. In order to be well formed, an operator must return 0 if any of its inputs are less that zero or outside its bounds. In listing D.1, we show the well formed-ness of the Mermin polynomials using induction on the size of the operators, we are furthermore doing this proof at the same time for M_n and M'_n (for the definition of the Mermin polynomials, see Sec. 2.4.1). The reader can see in this proof one difficulty of Coq proofs: each theorem must be explicitly used, for example, the fact that the Kronecker product of two well formed operators is a well formed operator (called by the command `apply WF_kron`) is quite trivial for a human, but not so for Coq.

The proof that Mermin polynomials are Hermitian is quite similar in spirit, the difference this time is that the calculation for the coefficients of the matrices must be perform, adding a layer of difficulty. But the proof is still quite compact thanks to the theorems and tactics implemented by the creators of SQIRE (the tactic `unify_pows_two` was a real life savior!).

```
Lemma WF_Mermin : ∀ a a' : nat → Square 2,
  (∀ (n : nat), (WF_Unitary (a n) ∧ WF_Unitary (a' n))) →
  (∀ (n : nat), WF_Matrix (Mermin_polynomial_Gen n a a') ∧
              WF_Matrix (Mermin_polynomial_Gen' n a a')).
Proof.
  intros a a' H n. case n.
  - { (* n = 0 *)
      split; unfold Mermin_polynomial_Gen, Mermin_polynomial_Gen'; apply WF_I.
    }
  - { (* n ≥ 1 *)
      intros n0. induction n0.
      - { (* n = 1 *)
          split; unfold Mermin_polynomial_Gen, Mermin_polynomial_Gen'; apply H.
        }
      - { (* n > 1 *)
          split.
          - { (* WF_Matrix (Mermin_polynomial_Gen (n1+2) a a') *)
              rewrite Mermin_pol_dev_once.
              replace (2 ^ S (S n0))%nat with (2 ^ S n0 * 2)%nat by
                unify_pows_two.
              apply WF_plus. apply WF_scale. apply WF_kron.
              reflexivity. reflexivity.
              apply IHn0. apply WF_plus. apply H. apply H. apply WF_scale.
              apply WF_kron.
              reflexivity. reflexivity.
              apply IHn0. apply WF_minus. apply H. apply H.
            }
          - { (* WF_Matrix (Mermin_polynomial_Gen' (n1+2) a a') *)
              rewrite Mermin_pol'_dev_once.
              replace (2 ^ S (S n0))%nat with (2 ^ S n0 * 2)%nat by
                unify_pows_two.
              apply WF_plus. apply WF_scale. apply WF_kron.
              reflexivity. reflexivity.
              apply IHn0. apply WF_plus. apply H. apply H. apply WF_scale.
              apply WF_kron.
              reflexivity. reflexivity.
              apply IHn0. apply WF_minus. apply H. apply H.
            }
        }
    }
Qed.
```

Listing D.1: Proofs of well formed-ness of Mermin polynomials in Coq

```
Lemma Mermin_Hermitian : ∀ a a' : nat → Square 2,
    (∀ (n : nat), (WF_Unitary (a n) ∧ Hermitian (a n) ∧
              WF_Unitary (a' n) ∧ Hermitian (a' n))) →
    (∀ (n : nat), Hermitian (Mermin_polynomial_Gen n a a') ∧
              Hermitian (Mermin_polynomial_Gen' n a a')).
Proof.
  unfold Hermitian. intros a a' H n. case n.
  - { (* n = 0 *)
      unfold Mermin_polynomial_Gen, Mermin_polynomial_Gen'.
      split; apply id_adjoint_eq.
    }
  - { (* n ≥ 1 *)
      intro n0.
      induction n0.
      - { (* n = 1 *)
          unfold Mermin_polynomial_Gen, Mermin_polynomial_Gen'.
          split; apply H.
        }
      - { (* n > 1 *)
          assert ((Mermin_polynomial_Gen (S n0) a a') † =
                  Mermin_polynomial_Gen (S n0) a a') as SAr.
          apply IHn0.
          assert ((Mermin_polynomial_Gen' (S n0) a a') † =
                  Mermin_polynomial_Gen' (S n0) a a') as SAr'.
          apply IHn0.
          split.
          - { (* Hermitian (Mermin_polynomial_Gen (n1 + 2) a a') *)
              rewrite Mermin_pol_dev_once.
              replace (2 ^ S (S n0))%nat with (2 ^ S n0 * 2)%nat by
                unify_pows_two.
              rewrite Mplus_adjoint. rewrite Mscale_adj. rewrite Mscale_adj.
              rewrite kron_adjoint. rewrite kron_adjoint. rewrite Mplus_adjoint.
              rewrite Mminus_adjoint.
              rewrite SAr'.
              rewrite SAr.
              destruct (H (S (S n0))) as [_ [SA_a [_ SA_a']]].
              rewrite SA_a, SA_a'.
              rewrite SA_half.
              reflexivity.
            }
          - { (* Hermitian (Mermin_polynomial_Gen' (n1 + 2) a a') *)
              rewrite Mermin_pol'_dev_once.
              replace (2 ^ S (S n0))%nat with (2 ^ S n0 * 2)%nat by
                unify_pows_two.
              rewrite Mplus_adjoint. rewrite Mscale_adj. rewrite Mscale_adj.
              rewrite kron_adjoint. rewrite kron_adjoint. rewrite Mplus_adjoint.
              rewrite Mminus_adjoint.
              rewrite SAr'.
              rewrite SAr.
              destruct (H (S (S n0))) as [_ [SA_a [_ SA_a']]].
              rewrite SA_a, SA_a'.
              rewrite SA_half.
              reflexivity.
            }
        }
    }
Qed.
```

Listing D.2: Proofs of Hermitian-ness of Mermin polynomials in Coq

Index

List of Figures

LIST OF TABLES

Bibliography

[AB11] Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13(11):113036, November 2011. https://doi.org/10.1088/1367-2630/13/11/113036.

[ABJ+20] Grâce Amouzou, Jeoffrey Boffelli, Hamza Jaffali, Kossi Atchonouglo, and Frédéric Holweck. Entanglement and non-locality of four-qubit connected hypergraph states. *arXiv:2010.03217 [math-ph, physics:quant-ph]*, October 2020. http://arxiv.org/abs/2010.03217.

[ACG+16] Daniel Alsina, Alba Cervera, Dardo Goyeneche, José I. Latorre, and Karol Życzkowski. Operational approach to Bell inequalities: Application to qutrits. *Physical Review A*, 94(3):032102, September 2016. https://link.aps.org/doi/10.1103/PhysRevA.94.032102.

[AL16] Daniel Alsina and José Ignacio Latorre. Experimental test of Mermin inequalities on a 5-qubit quantum computer. *Physical Review A*, 94(1), July 2016. http://arxiv.org/abs/1605.04220.

[ARBC09] Elias Amselem, Magnus Rådmark, Mohamed Bourennane, and Adán Cabello. State-independent quantum contextuality with single photons. *Physical Review Letters*, 103(16):160405, October 2009.

[Ark12] Alex Arkhipov. Extending and Characterizing Quantum Magic Games. *arXiv:1209.3819 [quant-ph]*, September 2012. http://arxiv.org/abs/1209.3819.

[Aul12] Martin Aulbach. Classification of entanglement in symmetric states. *International Journal of Quantum Information*, 10(07):1230004, October 2012. https://www.worldscientific.com/doi/abs/10.1142/S0219749912300045.

[Bac14] Miriam Backens. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, September 2014. https://doi.org/10.1088/1367-2630/16/9/093021.

[BBT05] Gilles Brassard, Anne Broadbent, and Alain Tapp. Quantum Pseudo-Telepathy. *Foundations of Physics*, 35(11):1877–1907, November 2005. https://doi.org/10.1007/s10701-005-7353-4.

[BC13] Francis Buekenhout and Arjeh M. Cohen. *Diagram Geometry: Related to Classical Groups and Buildings*. Springer Science & Business Media, January 2013.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 24(3):235–265, September 1997. http://www.sciencedirect.com/science/article/pii/S074771719690125X.

[BDL12] L. Borsten, M. J. Duff, and P. Lévay. The black-hole/qubit correspondence: An up-to-date review. *Classical and Quantum Gravity*, 29(22):224008, October 2012. https://doi.org/10.1088/0264-9381/29/22/224008.

[Bel66] John S. Bell. On the Problem of Hidden Variables in Quantum Mechanics. *Reviews of Modern Physics*, 38(3):447–452, July 1966. https://link.aps.org/doi/10.1103/RevModPhys.38.447.

[BGS05] Nicolas Brunner, Nicolas Gisin, and Valerio Scarani. Entanglement and nonlocality are different resources. *New Journal of Physics*, 7:88–88, April 2005. https://doi.org/10.1088%2F1367-2630%2F7%2F1%2F088.

[BI17] Lev S. Bishop and IBM Quantum Team. QASM 2.0: A Quantum Circuit Intermediate Representation. 2017:P46.008, March 2017. https://ui.adsabs.harvard.edu/abs/2017APS..MARP46008B.

[Bir67] Brian J Birch. Cyclotomic fields and Kummer extensions. In *Algebraic Number Theory (Proc. Instructional Conf., Brighton, 1965), Thompson, Washington, DC*, pages 85–93, 1967.

[BNO02] Ofer Biham, Michael A. Nielsen, and Tobias J. Osborne. Entanglement monotone derived from Grover's algorithm. *Physical Review A*, 65(6):062312, June 2002. https://link.aps.org/doi/10.1103/PhysRevA.65.062312.

[BOF$^+$16] J. Batle, C. H.Raymond Ooi, Ahmed Farouk, M. S. Alkhambashi, and S. Abdalla. Global versus local quantum correlations in the Grover search algorithm. *Quantum Information Processing*, 15(2):833–849, February 2016. http://link.springer.com/10.1007/s11128-015-1174-y.

[BP02] Samuel L. Braunstein and Arun K. Pati. Speed-up and Entanglement in Quantum Searching. *Quantum Info. Comput.*, 2(5):399–409, August 2002. http://dl.acm.org/citation.cfm?id=2011483.2011489.

[BRN$^+$20] Fraser Brown, John Renner, Andres Nötzli, Sorin Lerner, Hovav Shacham, and Deian Stefan. Towards a verified range analysis for JavaScript JITs.

In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 135–150, New York, NY, USA, June 2020. Association for Computing Machinery. https://doi.org/10.1145/3385412.3385968.

[Bro82] Allan G. Bromley. Charles Babbage's Analytical Engine, 1838. *Annals of the History of Computing*, 4(3):196–217, July 1982.

[Cab08] Adan Cabello. Experimentally testable state-independent quantum contextuality. *Physical Review Letters*, 101(21):210401, November 2008. http://arxiv.org/abs/0808.2456.

[Cab21] Adán Cabello. Bell Non-locality and Kochen–Specker Contextuality: How are They Connected? *Foundations of Physics*, 51(3):61, May 2021. https://doi.org/10.1007/s10701-021-00466-5.

[Cam92] Peter J. Cameron. *Projective and Polar Spaces*. University of London, Queen Mary and Westfield College, January 1992. http://www.maths.qmw.ac.uk/~pjc/pps/.

[CBAK13] Shantanav Chakraborty, Subhashish Banerjee, Satyabrata Adhikari, and Atul Kumar. Entanglement in the Grover's Search Algorithm. *arXiv:1305.4454 [quant-ph]*, May 2013. http://arxiv.org/abs/1305.4454.

[CBB$^+$21] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In Nobuko Yoshida, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 148–177, Cham, 2021. Springer International Publishing.

[Cd19] Jessy Colonval and Henri de Boutray. Formalisation et validation d'une méthode de construction de systèmes de blocs. In *18e journées Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 51–58, Toulouse, Fra, June 2019. https://hal.archives-ouvertes.fr/hal-02945084.

[CEG96] Adán Cabello, José M. Estebaranz, and Guillermo García-Alcaine. Bell-Kochen-Specker theorem: A proof with 18 vectors. *Physics Letters A*, 212(4):183–187, March 1996. http://www.sciencedirect.com/science/article/pii/037596019600134X.

[CGP$^+$02] Daniel Collins, Nicolas Gisin, Sandu Popescu, David Roberts, and Valerio Scarani. Bell-type inequalities to detect true n-body non-separability. *Physical Review Letters*, 88(17):170405, April 2002. http://arxiv.org/abs/quant-ph/0201058.

[CHSH69] John F. Clauser, Michael A. Horne, Abner Shimony, and Richard A. Holt. Proposed Experiment to Test Local Hidden-Variable Theories. *Physical Review Letters*, 23(15):880–884, October 1969. https://link.aps.org/doi/10.1103/PhysRevLett.23.880.

[CM13] Richard Cleve and Rajat Mittal. Characterization of Binary Constraint System Games. *arXiv:1209.2729 [quant-ph]*, October 2013. http://arxiv.org/abs/1209.2729.

[CM14] D. Cofer and Steven P. Miller. DO-333 Certification Case Studies. In *NASA Formal Methods*, 2014.

[Col10] Charles J. Colbourn. *CRC Handbook of Combinatorial Designs*. CRC Press, December 2010. https://www.taylorfrancis.com/books/9781420049954.

[Coo10] K. Coolsaet. The smallest split Cayley hexagon has two symplectic embeddings. *Finite Fields and Their Applications*, 16(5):380–384, September 2010. https://www.sciencedirect.com/science/article/pii/S1071579710000572.

[Cro18] Andrew Cross. The IBM Q experience and QISKit open-source quantum computing software. 2018:L58.003, January 2018. https://ui.adsabs.harvard.edu/abs/2018APS..MARL58003C.

[dB21] Henri de Boutray. Quantcert. https://quantcert.github.io/, May 2021.

[Deu85] David Deutsch. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, July 1985. http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.1985.0070.

[dHGM21a] Henri de Boutray, Frédéric Holweck, Alain Giorgetti, and Pierre-Alain Masson. Automated detection of contextuality proofs with intermediate numbers of observables. In *18th International Conference on Quantum Physics and Logic*, Poster, Gdańsk (Poland), August 2021. https://qpl2021.eu/.

[dHGM21b] Henri de Boutray, Frédéric Holweck, Alain Giorgetti, and Pierre-Alain Masson. Automated synthesis of contextuality proofs from subspaces of symplectic polar spaces. *arXiv:2105.13798 [math-ph, physics:quant-ph]*, May 2021. http://arxiv.org/abs/2105.13798.

[Dij75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18:453–457, 1975.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A:*

Mathematical and Physical Sciences, 439(1907):553–558, December 1992. https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1992.0167.

[dJH+20] Henri de Boutray, Hamza Jaffali, Frédéric Holweck, Alain Giorgetti, and Pierre-Alain Masson. Non-locality and entanglement detection with Mermin polynomials for Grover's algorithm and the Quantum Fourier Transform. In *11th Colloquium of the CNRS GDR N°3322 on Quantum Engineering, Foundations & Applications, Ingénierie Quantique, Des Aspects Fondamentaux Aux Applications (GDR IQFA 2020)*, Poster, pages 27–27, Online (Electronic Conference), France, December 2020. https://publiweb.femto-st.fr/tntnet/entries/17460/documents/author/data.

[dJH+21] Henri de Boutray, Hamza Jaffali, Frédéric Holweck, Alain Giorgetti, and Pierre-Alain Masson. Mermin polynomials for non-locality and entanglement detection in Grover's algorithm and Quantum Fourier Transform. *Quantum Information Processing*, 20(3):91, March 2021. https://doi.org/10.1007/s11128-020-02976-z.

[DRLB20] A. Dikme, N. Reichel, A. Laghaout, and G. Björk. Measuring the Mermin-Peres magic square using an online quantum computer. *arXiv:2009.10751 [quant-ph]*, September 2020. http://arxiv.org/abs/2009.10751.

[EJ98] Artur Ekert and Richard Jozsa. Quantum algorithms: Entanglement-enhanced information processing. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, August 1998. https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1998.0248.

[ER88] P. H. Eberhard and R. R. Ross. Quantum Field Theory Cannot Provide Faster-Than-Light Communication. *Foundations of Physics Letters*, 2, May 1988. https://escholarship.org/uc/item/5604n7md.

[Fel21] Sophie Felix. French research at the heart of the Quantum Plan. https://news.cnrs.fr/articles/french-research-at-the-heart-of-the-quantum-plan, February 2021.

[Flo67] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967. http://laser.cs.umass.edu/courses/cs521-621.Spr06/papers/Floyd.pdf.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico

Tassi, Laurent Théry, and Sandrine Blazy. A Machine-Checked Proof of the Odd Order Theorem. In *Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013. http://link.springer.com/10.1007/978-3-642-39634-2_14.

[GKS+21] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. Quantum Computing: A Taxonomy, Systematic Review and Future Directions. *arXiv:2010.15559 [cs]*, April 2021. http://arxiv.org/abs/2010.15559.

[Gre14] Joel Greenberg. *Gordon Welchman: Bletchley Park's Architect of Ultra Intelligence*. Frontline Books, February 2014.

[Gro96] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, May 1996. ACM. http://doi.acm.org/10.1145/237814.237866.

[GT09] Otfried Gühne and Géza Tóth. Entanglement detection. *Physics Reports*, 474(1):1–75, April 2009. http://www.sciencedirect.com/science/article/pii/S0370157309000623.

[GW14] Gilad Gour and Nolan R. Wallach. On symmetric SL-invariant polynomials in four qubits. In Roger Howe, Markus Hunziker, and Jeb F. Willenbring, editors, *Symmetry: Representation Theory and Its Applications: In Honor of Nolan R. Wallach*, Progress in Mathematics, pages 259–267. Springer, New York, NY, 2014. https://doi.org/10.1007/978-1-4939-1590-3_9.

[HJN16] Frédéric Holweck, Hamza Jaffali, and Ismaël Nounouh. Grover's algorithm and the secant varieties. *Quantum Information Processing*, 15(11):4391–4413, November 2016. http://link.springer.com/10.1007/s11128-016-1445-2.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969. https://doi.org/10.1145/363235.363259.

[Hol19a] Frédéric Holweck. Geometric constructions over $\mathbb{C}$ and $\mathbb{F}_2$ for Quantum Information. In *Quantum Physics and Geometry*, volume 25 of *Lecture Notes of the Unione Matematica Italiana*, pages 87–124. Springer Nature, March 2019. http://arxiv.org/abs/1810.04258.

[Hol19b] Frédéric Holweck. *On the Projective Geometry of Entanglement and Contextuality*. Habilitation à diriger des recherches, Université Bourgogne Franche-Comté, September 2019. https://hal.archives-ouvertes.fr/tel-02913351.

[Hol21] Frédéric Holweck. Testing Quantum Contextuality of Binary Symplectic Polar Spaces on a Noisy Intermediate Scale Quantum Computer. *arXiv:2101.03812 [math-ph, physics:quant-ph]*, January 2021. http://arxiv.org/abs/2101.03812.

[HOS09] Hans Havlicek, Boris Odehnal, and Metod Saniga. Factor-Group-Generated Polar Spaces and (Multi-)Qudits. *Symmetry, Integrability and Geometry: Methods and Applications*, 5(096), October 2009. http://www.emis.de/journals/SIGMA/2009/096/.

[HRH⁺21] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for Quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):37:1–37:29, January 2021. https://doi.org/10.1145/3434318.

[HS00] A. Higuchi and A. Sudbery. How entangled can two couples get? *Physics Letters A*, 273(4):213–217, August 2000. http://www.sciencedirect.com/science/article/pii/S0375960100004801.

[HS17] Frédéric Holweck and Metod Saniga. Contextuality with a Small Number of Observables. *International Journal of Quantum Information*, 15(04):1750026, June 2017. http://arxiv.org/abs/1607.07567.

[HT16] J. W. P. Hirschfeld and J. A. Thas. *General Galois Geometries*. Springer Monographs in Mathematics. Springer-Verlag, London, 2016. https://www.springer.com/us/book/9781447167884.

[HWVE14] Mark Howard, Joel J. Wallman, Victor Veitch, and Joseph Emerson. Contextuality supplies the magic for quantum computation. *Nature*, 510(7505):351–355, June 2014. http://arxiv.org/abs/1401.4174.

[IBM] IBM Quantum. https://quantum-computing.ibm.com/.

[INT99] INTERNATIONAL ATOMIC ENERGY AGENCY. *Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control*. Number 384 in Technical Reports Series. Vienna, 1999. https://www.iaea.org/publications/5718/verification-and-validation-of-software-related-to-nuclear-power-plant-instrumentation-and-control.

[JH19] Hamza Jaffali and Frédéric Holweck. Quantum Entanglement involved in Grover's and Shor's algorithms: The four-qubit case. *Quantum Information Processing*, 18(5):133, May 2019. http://arxiv.org/abs/1811.08894.

[JL03] Richard Jozsa and Noah Linden. On the role of entanglement in quantum computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, August 2003. https://royalsocietypublishing.org/doi/10.1098/rspa.2002.1097.

[KM02] Jennifer D. Key and Jamshid 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. https://www.researchgate.net/publication/266514055_Codes_designs_and_graphs_from_the_Janko_groups_J_1_and_J_2.

[KM06] Vivien M. Kendon and William J. Munro. Entanglement and Its Role in Shor's Algorithm. *Quantum Info. Comput.*, 6(7):630–640, November 2006. http://dl.acm.org/citation.cfm?id=2011698.2011704.

[KM08] Jennifer D. Key and Jamshid 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.

[KS67] Simom Kochen and Ernst Specker. The Problem of Hidden Variables in Quantum Mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967. https://www.jstor.org/stable/24902153.

[Kv20] Aleks Kissinger and John van de Wetering. Reducing T-count with the ZX-calculus. *Physical Review A*, 102(2):022406, August 2020. http://arxiv.org/abs/1903.10477.

[KZG$^+$09] G. Kirchmair, F. Zähringer, R. Gerritsma, M. Kleinmann, O. Gühne, A. Cabello, R. Blatt, and C. F. Roos. State-independent experimental test of quantum contextuality. *Nature*, 460(7254):494–497, July 2009. http://arxiv.org/abs/0904.1655.

[LH19] Péter Lévay and Frédéric Holweck. Finite geometric toy model of spacetime as an error correcting code. *Physical Review D*, 99(8):086015, April 2019. https://link.aps.org/doi/10.1103/PhysRevD.99.086015.

[LHS17] Péter Lévay, Frédéric Holweck, and Metod Saniga. Magic three-qubit Veldkamp line: A finite geometric underpinning for form theories of gravity and black hole entropy. *Physical Review D*, 96(2):026018, July 2017. https://link.aps.org/doi/10.1103/PhysRevD.96.026018.

[LMP03] C. Lavor, L. R. U. Manssur, and R. Portugal. Grover's Algorithm: Quantum Database Search. *arXiv:quant-ph/0301079*, January 2003. http://arxiv.org/abs/quant-ph/0301079.

[LS17] Péter Lévay and Zsolt Szabó. Mermin pentagrams arising from Veldkamp lines for three qubits. *Journal of Physics A: Mathematical and Theoretical*, 50(9):095201, January 2017. https://doi.org/10.1088/1751-8121/aa56aa.

[LSP$^+$07] Thierry Lecomte, Thierry Servat, Guilhem Pouzancre, et al. Formal methods in safety-critical railway systems. In *10th Brasilian Symposium on Formal Methods*, pages 29–31, 2007. http://deploy-eprints.ecs.soton.ac.uk/8/1/fm_sc_rs_v2.pdf.

[LSVP09] Péter Lévay, Metod Saniga, Péter Vrana, and Petr Pracna. Black hole entropy and finite geometry. *Physical Review D*, 79(8):084036, April 2009. https://link.aps.org/doi/10.1103/PhysRevD.79.084036.

[LT03] J.-G. Luque and J.-Y. Thibon. The polynomial invariants of four qubits. *Physical Review A*, 67(4):042303, April 2003. http://arxiv.org/abs/quant-ph/0212069.

[Mat21] David Matthews. How to get started in quantum computing. *Nature*, 591(7848):166–167, March 2021. https://www.nature.com/articles/d41586-021-00533-x.

[Mer90] N David Mermin. Extreme quantum entanglement in a superposition of macroscopically distinct states. *Physical Review Letters*, 65(15):1838–1840, October 1990. https://link.aps.org/doi/10.1103/PhysRevLett.65.1838.

[Mer93] N. David Mermin. Hidden variables and the two theorems of John Bell. *Reviews of Modern Physics*, 65(3):803–815, July 1993. https://link.aps.org/doi/10.1103/RevModPhys.65.803.

[MLL19] Andrey Mokhov, Georgy Lukyanov, and Jakob Lechner. Formal verification of spacecraft control programs (experience report). In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell*, Haskell 2019, pages 139–145, New York, NY, USA, August 2019. Association for Computing Machinery. https://doi.org/10.1145/3331545.3342593.

[MW02a] David A. Meyer and Nolan R. Wallach. Global entanglement in multiparticle systems. *Journal of Mathematical Physics*, 43(9):4273–4278, August 2002. https://aip.scitation.org/doi/abs/10.1063/1.1497700.

[MW02b] Akimasa Miyake and Miki Wadati. Multipartite Entanglement and Hyperdeterminants. *Quantum Info. Comput.*, 2(7):540–555, December 2002. http://dl.acm.org/citation.cfm?id=2011499.2011503.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. www.cambridge.org.

[OS06] Andreas Osterloh and Jens Siewert. Entanglement monotones and maximally entangled states in multipartite qubit systems. *International Journal of Quantum Information*, 04(03):531–540, June 2006. https://www.worldscientific.com/doi/abs/10.1142/S0219749906001980.

[Per90] Asher Peres. Incompatible results of quantum measurements. *Physics Letters A*, 151(3):107–108, December 1990. https://www.sciencedirect.com/science/article/pii/037596019090172K.

[PGHS15] Michel Planat, Alain Giorgetti, Frédéric Holweck, and Metod Saniga. Quantum contextual finite geometries from dessins d'enfants. *International Journal of Geometric Methods in Modern Physics*, 12(07):1550067, March 2015. https://www.worldscientific.com/doi/abs/10.1142/S021988781550067X.

[Pla12] Michel Planat. On small proofs of the Bell-Kochen-Specker theorem for two, three and four qubits. *The European Physical Journal Plus*, 127(8):86, August 2012. https://doi.org/10.1140/epjp/i2012-12086-x.

[Por21] Jon Porter. Google wants to build a useful quantum computer by 2029. https://www.theverge.com/2021/5/19/22443453/google-quantum-computer-2029-decade-commercial-useful-qubits-quantum-transistor, May 2021.

[Pos36] Emil L. Post. Finite Combinatory Processes-Formulation 1. *The Journal of Symbolic Logic*, 1(3):103–105, 1936. https://www.jstor.org/stable/2269031.

[Pre20] Press release. £70 million funding to secure UK position as a world-leader in quantum technology. https://www.gov.uk/government/news/70-million-funding-to-secure-uk-position-as-a-world-leader-in-quantum-technology, June 2020.

[PS08] Michel Planat and Metod Saniga. On the Pauli graphs on N-qudits. *Quantum Information & Computation*, 8(1):127–146, January 2008.

[PSH13] Michel Planat, Metod Saniga, and Frédéric Holweck. Distinguished three-qubit 'magicity' via automorphisms of the split Cayley hexagon. *Quantum Information Processing*, 12(7):2535–2549, July 2013. https://doi.org/10.1007/s11128-013-0547-3.

[PSVM01] Burkard Polster, Andreas E. Schroth, and Hendrik Van Maldeghem. Generalized Flatland. *The Mathematical Intelligencer*, 23(4):33–47, September 2001. https://doi.org/10.1007/BF03024601.

[Qua] Azure Quantum. Azure Quantum - Quantum Service | Microsoft Azure. https://azure.microsoft.com/en-us/services/quantum/.

[RBM13] M. Rossi, D. Bruß, and C. Macchiavello. Scale invariance of entanglement dynamics in Grover's quantum search algorithm. *Physical Review A - Atomic, Molecular, and Optical Physics*, 87(2):1–5, 2013.

[SAC+06] K.M. Svore, A.V. Aho, A.W. Cross, I. Chuang, and I.L. Markov. A layered software architecture for quantum computing design tools. *Computer*, 39(1):74–83, January 2006.

[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/icerm_report.pdf.

[SdHG21] Metod Saniga, Henri de Boutray, Frédéric Holweck, and Alain Giorgetti. Taxonomy of Polar Subspaces of Multi-Qubit Symplectic Polar Spaces of Small Rank. *Mathematics*, 9(18):2272, September 2021. https://www.mdpi.com/2227-7390/9/18/2272.

[SGL+10] Metod Saniga, Richard M. Green, Péter Lévay, Petr Pracna, and Peter Vrana. The Veldkamp Space of GQ(2,4). *International Journal of Geometric Methods in Modern Physics*, 07(07):1133–1145, November 2010. http://arxiv.org/abs/0903.0715.

[Shi95] Abner Shimony. Degree of Entanglement. *Annals of the New York Academy of Sciences*, 755(1):675–679, 1995. https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1995.tb39008.x.

[SHJ20] Metod Saniga, Frédéric Holweck, and Hamza Jaffali. Taxonomy of Three-Qubit Mermin Pentagrams. *Symmetry*, 12(4):534, April 2020. https://www.mdpi.com/2073-8994/12/4/534.

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, 1994. IEEE Comput. Soc. Press. http://ieeexplore.ieee.org/document/365700/.

[SL12] Michel Saniga and Péter Lévay. Mermin's pentagram as an ovoid of PG(3,2). *EPL (Europhysics Letters)*, 97(5):50006, March 2012. https://doi.org/10.1209/0295-5075/97/50006.

[SLPP09] Metod Saniga, Péter Lévay, Michel Planat, and Petr Pracna. Geometric Hyperplanes of the Near Hexagon L3 $\times$ GQ(2, 2). *Letters in Mathematical Physics*, 91(1):93, October 2009. https://doi.org/10.1007/s11005-009-0362-z.

[Smi18] Lamar Smith. Text - H.R.6227 - 115th Congress (2017-2018): National Quantum Initiative Act. https://www.congress.gov/bill/115th-congress/house-bill/6227/text, December 2018.

[SP07] Metod Saniga and Michel Planat. Multiple Qubits as Symplectic Polar Spaces of Order Two. *Advanced Studies in Theoretical Studies*, 1(1):1–4, 2007. http://arxiv.org/abs/quant-ph/0612179.

[SPPH07] Metod Saniga, Michel Planat, Petr Pracna, and Hans Havlicek. The Veldkamp Space of Two-Qubits. *SIGMA. Symmetry, Integrability and Geometry: Methods and Applications*, 3:075, June 2007. http://www.emis.de/journals/SIGMA/2007/075/.

[SSB05] Yishai Shimoni, Daniel Shapira, and Ofer Biham. Entangled Quantum States Generated by Shor's Factoring Algorithm. *Physical Review A*, 72(6):062308, December 2005. http://arxiv.org/abs/quant-ph/0510042.

[TG05] Geza Toth and Otfried Guehne. Entanglement Detection in the Stabilizer Formalism. *Physical Review A*, 72(2):022340, August 2005. http://arxiv.org/abs/quant-ph/0501020.

[Tha09] K. Thas. The geometry of generalized Pauli operators of N-qudit Hilbert space, and an application to MUBs. *EPL (Europhysics Letters)*, 86(6):60005, June 2009. https://iopscience.iop.org/article/10.1209/0295-5075/86/60005.

[UL07] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. January 2007.

[VDDMV02] F. Verstraete, J. Dehaene, B. De Moor, and H. Verschelde. Four qubits can be entangled in nine different ways. *Physical Review A*, 65(5):052112, April 2002. https://link.aps.org/doi/10.1103/PhysRevA.65.052112.

[Vid00] Guifré Vidal. Entanglement monotones. *Journal of Modern Optics*, 47(2-3):355–376, February 2000. https://www.tandfonline.com/doi/abs/10.1080/09500340008244048.

[VL10] Péter Vrana and Péter Lévay. The Veldkamp space of multiple qubits. *Journal of Physics A: Mathematical and Theoretical*, 43(12):125303, March 2010. https://doi.org/10.1088/1751-8113/43/12/125303.

[VPL19] Nathalie Vidal, Aurélie Del Prete, and Frédéric Laurent. Expériences et investigations autour des machines arithmétiques de Blaise Pascal. page 13, 2019.

[WA11] Mordecai Waegell and P. K. Aravind. Parity Proofs of the Kochen-Specker Theorem Based on the 24 Rays of Peres. *Foundations of Physics*, 41(12):1786–1799, December 2011. https://doi.org/10.1007/s10701-011-9578-8.

[WG03] Tzu-Chieh Wei and Paul M. Goldbart. Geometric measure of entanglement and applications to bipartite and multipartite quantum states. *Physical Review A*, 68(4):042307, October 2003. https://link.aps.org/doi/10.1103/PhysRevA.68.042307.

[Yin11] Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):1–49, June 2011. http://dl.acm.org/citation.cfm?doid=2049706.2049708.

[Yin18] Mingsheng Ying. Toward automatic verification of quantum programs. *Formal Aspects of Computing*, 31(1):3–25, February 2018. https://doi.org/10.1007/s00165-018-0465-3.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the `tex-upmethodology` package suite — http://www.arakhne.org/tex-upmethodology/

SPIM

école doctorale **sciences pour l'ingénieur et microtechniques**

Title: Computational studies of entanglement and quantum contextuality properties towards their formal verification

Keywords: Quantum computing, Entanglement, Contextuality, Formal specification, Verification

Abstract:

Although current quantum computers are limited to the use of a few quantum bits, the foundations of quantum programing have been growing over the last 20 years. These foundations have been theorized as early as in the 80's but the complexity of their implementation caused these leads to be out of reach until very recently. In this context, the objective of this thesis is to contribute to the adaptation of the methods of formal specification and deductive verification of classical programs to quantum programs. I thus present my contributions to the study of quantum properties with the end goal of formalizing them. I study in particular quantum entanglement and quantum contextuality. These properties allow to classify quantum states and protocols, and in particular to differentiate them from classical ones. My study of entanglement is based more specifically on the evolution of entanglement during two quantum algorithms: the Grover algorithm and the Quantum Fourier Transform. To quantify entanglement along those algorithms, I use Mermin's polynomials, which have the advantage of being implementable in actual quantum computers. My study of contextuality, on the other hand, relies on finite geometries representing experiments, which are said to be contextual when no non-contextual classical theory can predict the results. These geometries are associated with the binary symplectic polar spaces. We study their structure, and eventually use this structure to get insights on quantum protocols using contextuality. The study of these properties led to interesting conjectures which we started to formalize in proof environments, such as Coq and Why3, but are left as perspective as these works have not reach a conclusion yet.

Titre : Études calculatoires de l'intrication et de la contextualité quantiques dans la perspective de leur vérification formelle

Mots-clés : Informatique quantique, Intrication, Contextualité, Spécification formelle, Vérification

Résumé :

Bien que les ordinateurs quantiques actuels se limitent à l'utilisation de quelques qubits, les fondations de la programmation quantique se sont développées au cours des 20 dernières années. Ces fondations ont été théorisées dès les années 80 mais la complexité de leur mise en œuvre a rendu ces pistes inaccessibles jusqu'à très récemment. Dans ce contexte, l'objectif de cette thèse est de contribuer à l'adaptation des méthodes de spécification formelle et de vérification déductive des programmes classiques aux programmes quantiques. Je présente donc mes contributions à l'étude des propriétés quantiques dans le but de les formaliser. J'étudie en particulier l'intrication quantique et la contextualité quantique. Ces propriétés permettent de classifier les états et les protocoles quantiques, et en particulier de les différencier des états et protocoles classiques. Mon étude de l'intrication est fondée plus spécifiquement sur l'évolution de l'intrication au cours de deux algorithmes quantiques : l'algorithme de Grover et la transformée de Fourier quantique. Pour quantifier l'intrication le long de ces algorithmes, j'utilise des polynômes de Mermin, qui ont l'avantage de pouvoir être implémentables dans des ordinateurs quantiques réels. Mon étude de la contextualité s'appuie, elle, sur des géométries finies représentant des expériences, qui sont dites contextuelles quand aucune théorie classique non-contextuelle ne peut en prédire les résultats. Ces géométries sont associées à des espaces polaires symplectiques binaires dont nous explorons la structure, avec comme objectif d'utiliser cette structure pour obtenir des intuitions sur les protocoles quantiques en utilisant la contextualité. L'étude de ces propriétés a conduit à des conjectures intéressantes que nous avons commencé à formaliser dans des environnements de preuve tels que Coq et Why3, mais qui sont laissées en perspective car ces travaux n'ont pas encore abouti.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon, France