CentraleSupélec

université PARIS-SACLAY

Autour des Diagrammes de Décision Quantiques

Rapport – Parcours recherche – Projet S7

Élève : Malo Leroy
Encadrant : Renaud Vilmart

Table des matières

17 février 2025

Chapitre 1

Introduction

1.1 Contexte et objectif

L'**informatique quantique** est un domaine en plein essor. Cette technologie permettant de manipuler des qubits à la place des bits qui sont à la base de l'informatique actuelle, ouvrent la voie à des algorithmes plus performants que les algorithmes classiques. Les machines quantiques exécutant ces algorithmes étant encore en développement et d'un coût important, il existe un besoin d'outils de simulation et de vérification d'algorithmes quantiques par des machines classiques. L'objectif de ce projet est de proposer un modèle de la structure de données des diagrammes de décision quantiques additifs abstraits, en s'appuyant sur les travaux existants, et de les simuler pour en étudier les performances.

1.2 État de l'art

Informatique quantique

Le premier postulat de la physique quantique, le **principe de superposition**, énonce que l'espace des états d'un système quantique est un espace de Hilbert. Par conséquent, si un *bit* ne peut classiquement être que dans un état $|0\rangle$ ou $|1\rangle$, son homologue quantique, le **qubit**, peut être dans une superposition de ces deux états.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

où α, β sont des coefficients complexes. Le second postulat, le *principe de mesure*, stipule que lorsqu'un qubit est mesuré, il est projeté sur un des états de base $|0\rangle$ ou $|1\rangle$ avec une probabilité $|\alpha|^2$ ou $|\beta|^2$ respectivement.

Les états à n qubits peuvent être représentés par des **vecteurs** de dimension 2^n, les états de l'ensemble constitué de deux systèmes étant ceux obtenus par produit tensoriel (*produit de Kronecker*) d'un état du premier système et d'un état du second. C'est ce nombre exponentiel de paramètres complexes pour un état à plusieurs qubits qui rend l'étude classique d'algorithmes quantiques difficile, puisque les états prennent une taille exponentiellement grande en mémoire.

Comme en informatique classique, les opérations élémentaires sur la mémoire sont réalisées en informatique quantique par des **portes**. D'un point de vue mathématique, une porte opérant sur n qubits est couramment représentée par une matrice de taille $2^n \times 2^n$. Appliquer une porte M à un ensemble de qubits v revient alors à multiplier le vecteur d'état par la matrice de la porte.

L'application en parallèle de plusieurs portes sur plusieurs qubits est représentée par le **produit tensoriel** (produit de Kronecker) des matrices de portes. On note toutefois que l'application de cette manière de portes sur des qubits se fait en un temps exponentiel en fonction du nombre de qubits, ce qui rend l'utilisation naïve d'algorithmes quantiques sur des machines classiques inefficace.

L'application de portes à des qubits est fréquemment représentée sous forme de **circuits quantiques**, où les qubits sont représentés par des lignes et les portes par des boîtes.

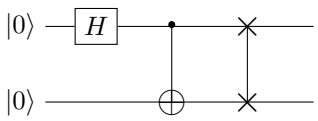

FIGURE 1.1 – Circuit à 2 qubits avec les portes H, CX et S

N'importe quelle matrice *unitaire* et *réversible* peut être utilisée comme porte quantique. Parmi les portes les plus courantes, on compte la porte de Hadamard (H), la porte « non » (X), la porte « non contrôlée » (CX), ou la porte d'échange (S pour *swap*). Ces portes sont avec d'autres utilisées pour réaliser des algorithmes quantiques, comme l'algorithme de Deutsch-Josza [1], l'algorithme de Grover [2], ou encore celui de Shor [3].

Afin de proposer un langage unifié permettant de développer des algorithmes quantiques à partir de portes pouvant être interprété sur n'importe quel matériel, IBM a publié en 2017 le langage de programmation **Open QASM**. [4] On y définit des qubits sur lesquels on applique des portes, et on peut ensuite simuler ou exécuter le programme sur un ordinateur quantique comme sur un ordinateur classique. Le circuit précédent peut par exemple être représenté par le programme 1.2. :

```
qubit a;
qubit b;
h a;
cx a b;
s a b;
```

Programme 1.2 – Exemple de code Open QASM

Diagrammes de décision

La structure de **diagramme de décision** est une structure de données développée à partir des années 1970. Elle est depuis devenue largement utilisée en informatique, notamment pour rendre plus compacte la représentation de **fonctions binaires** [5].

Prenons par exemple la fonction binaire $f(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge x_3)$. La représentation d'une telle fonction se fait dans le cas général à l'aide d'une *table de vérité*, donc dans une taille grandissant exponentiellement par rapport au nombre de variables binaires considéré. On peut utiliser une représentation plus compacte en utilisant un diagramme de décision, comme le montre la figure 1.3, où les fils gauches sont signalés par des flèches pointillées et les fils droits sont signalés par des flèches continues.

Les diagrammes de décision prennent avantage de la **structure** interne de la donnée (ici, une fonction booléenne). On note d'une part que les labels ne sont pas nécessaires pour reconstituer les valeurs prises par la fonction. D'autre part, dans le pire des cas, c'est-à-dire celui où la fonction ne possède aucune structure permettant une réduction, la taille du diagramme de décision (son nombre de branches) vaut $2^{n+1} - 2$ ce qui comme le nombre de valeurs 2^n à stocker dans une table de vérité, est **exponentiel** en n. Dans le pire cas, les diagrammes de décision ne permettent pas d'amélioration, mais ne sont pas non plus asymptotiquement pire que les tables de vérité.

Interprétation abstraite

L'interprétation abstraite est une méthode générale consistant à traiter des propriétés de programmes informatiques en les abstrayant. Elle a été introduite par Patrick et Radhia Cousot en 1977 [6]. Elle peut aussi être utilisée pour résoudre des problèmes ou calculer plus rapidement.

Étudions le problème suivant : on cherche à déterminer le signe de l'expression $-12 \times 7 - 13$. On pourrait calculer le résultat de cette expression, mais on peut aussi remarquer que -12 et -13 sont négatifs et que 7 est positif, d'où -12×7 négatif, ainsi l'expression est négative. Plus formellement, on est parti des **éléments concrets** que sont -12, 7 et -13 et on les a remplacés par les **éléments abstraits** « positif » $\oplus$ et « négatif » $\ominus$, sur lesquels on définit les opérations de somme et de produit selon les règles bien connues.

x_1	x_2	x_3	$f(x_1, x_2)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

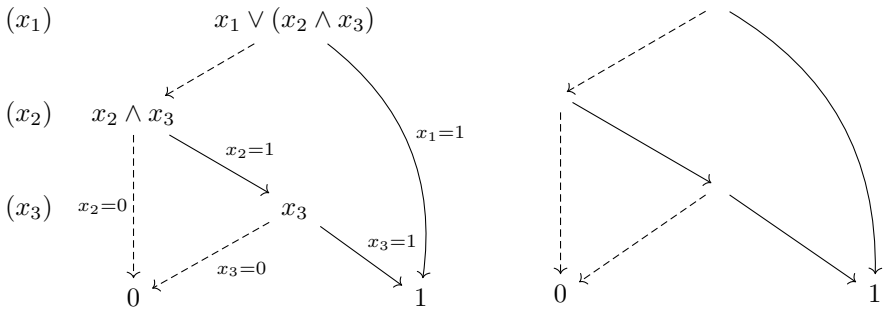

Figure 1.3 – (a) Table de vérité (b) Diagramme de décision (c) Diagramme de décision sans labels

Il existe de nombreux cas d'utilisation de l'interprétation abstraite [7], pour la compilation de programmes par exemple. Nous l'utiliserons dans ce projet pour simplifier les diagrammes de décision quantiques quitte à perdre une partie de l'information qu'ils contiennent. C'est aussi le cas pour l'exemple précédent : abstraire les éléments par leur signe ne permet pas toujours de déterminer le signe de l'expression.

1.3 Contribution

Les différents concepts présentés dans l'état de l'art permettent tous d'améliorer la vitesse de calcul ou la taille en mémoire d'une donnée. Il s'agit donc de **combiner ces concepts** dans ce projet, en ajoutant une dimension supplémentaire, l'additivité : le fait pour un diagramme d'avoir plusieurs fils droits (respectivement gauches), l'interprétation étant que le « fils droit effectif » est la somme des fils droits (respectivement gauches) du diagramme. L'objectif du projet était donc de proposer un modèle de diagrammes de décision quantiques abstraits et additifs, et de les simuler pour en étudier les performances en comparaison d'autres modèles.

De manière préliminaire, une étude des intervalles complexes polaires a été réalisée, ce qui n'avait pas été fait jusqu'à présent. Un **modèle** mathématique de la structure de données et de méthodes de réduction ont été formalisés, et des algorithmes de réduction ont été développés pour simplifier ces diagrammes. De plus, une des méthodes d'application de portes quantiques aux diagrammes ont été formalisées. Une **implémentation** de ces algorithmes a été réalisée, sans se baser sur des bibliothèques existantes (mis à part pour les tests unitaires).

1.4 Structure du rapport

Le chapitre 2 présente le modèle théorique de diagrammes de décision quantiques qui a été développé, y compris les algorithmes de réduction et d'application de portes qui accompagnent celui-ci. Ensuite, le chapitre 3 discute de l'implémentation réalisée. Finalement, une courte conclusion est présentée dans le chapitre 4.

Un document théorique plus complet que ce rapport destiné à être publié et rédigé en anglais, est disponible en ligne. D'éventuels points théoriques succinctement présentés dans ce rapport et ou théorèmes dont la preuve n'est pas spécifiée sont détaillés dans ce document. [8] De plus, une documentation technique du code est disponible en ligne. [9]

Chapitre 2

Modèle théorique

Le modèle théorique proposé est celui des **diagrammes de décision quantiques additifs abstraits**.

2.1 Arithmétique des intervalles

L'arithmétique des intervalles dans le cadre de l'interprétation abstraite est proche de l'exemple de calcul sur les signes de la section 1.2. L'arithmétique des intervalles est utilisée pour déterminer un ensemble de valeurs possibles pour une expression mathématique en utilisant des intervalles pour les valeurs des variables.

L'arithmétique des intervalles réels a été étudiée [10], et l'arithmétique des intervalles complexes cartésien a fait l'objet d'études légères par le passé [11]. Au cours de ce projet, un travail a été réalisé pour explorer les possibilités de l'arithmétique des **intervalles complexes cartésiens et polaires**.

Les intervalles complexes cartésiens sont définis par un intervalle pour la partie réelle et un intervalle pour la partie imaginaire. De manière équivalente, ils sont définis comme le plus petit rectangle dans le plan complexe (orienté selon les axes réel et imaginaire) contenant deux complexes. Les intervalles complexes polaires sont définis par un intervalle pour le module et un intervalle pour l'argument. Ces deux types d'intervalles ont des représentations différentes dans le plan complexe, comme le montre la figure 2.1. Dans un cas comme dans l'autre, l'équivalent abstrait d'une **opération** $*$ est défini sur des éléments α et β de l'ensemble des intervalles cartésiens $\mathcal{A}_0$ ou polaires $\mathcal{S}_0$ par

$$\alpha * \beta = \bigcap_{\gamma \supset \alpha \circledast \beta \text{ et } \gamma \in \mathcal{A}_0} \gamma \quad \text{où} \quad \alpha \circledast \beta = \{a * b; a \in \alpha, b \in \beta\}$$

Les opérations de somme, de produit et d'union ainsi construites sont **sur-approximées** : elles garantissent que l'ensemble des valeurs possibles est inclus dans l'ensemble abstrait, et d'avoir un intervalle pour résultat. Ces opérations ont des propriétés, de distributivité par exemple, parfois très différentes des nombres complexes qu'ils représentent. Des propriétés sur les intervalles cartésiens et polaires sont énoncées et démontrées dans le document annexe [8].

D'un point de vue pratique, les intervalles cartésiens sont généralement plus simples à manipuler que les intervalles polaires, et sont largement plus adaptés à une structure additive. Les intervalles polaires ont des avantages pour les opérations de multiplication et de division, mais rendent la somme coûteuse en calculs et en perte de précision.

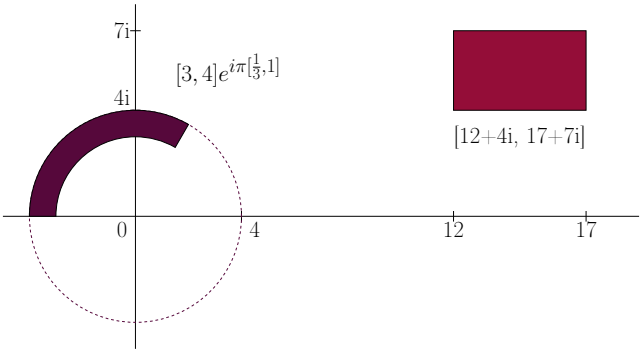

FIGURE 2.1 – Exemple d'intervalles complexes cartésiens et polaires

2.2 États abstraits et diagrammes

Les **états** abstraits à n qubits sont définis comme des 2^n-uplets d'intervalles complexes, on note leur ensemble $\mathcal{A}_n$. Il peut s'agit des intervalles cartésiens ou polaires, puisque les opérations sont définies de manière similaire, mais en pratique on se limite souvent aux intervalles cartésiens. On définit sur les états la relation d'ordre d'inclusion des états par l'inclusion des produits cartésiens des intervalles les composant.

Les **diagrammes** sont définis de manière récursive. Le seul diagramme de hauteur 0 est $\boxed{1}$, puis si l'ensemble $\mathcal{D}_n$ des diagrammes de hauteur n est défini, les diagrammes de hauteur $n+1$ peuvent avoir un nombre fini de fils gauches dans $\mathcal{D}_n$ et un nombre fini de fils droits dans $\mathcal{D}_n$, chacun étant associé à une amplitude abstraite sur la branche dans $\mathcal{A}_0$

$$\mathcal{D}_{n+1} = \mathscr{P}_f(\mathcal{A}_0 \times \mathcal{D}_n) \times \mathscr{P}_f(\mathcal{A}_0 \times \mathcal{D}_n)$$

On peut ainsi définir la fonction d'évaluation $\mathcal{E} : \mathcal{D}_n \to \mathcal{A}_n$ sur les diagrammes (une définition similaire est possible en utilisant les intervalles polaires), ce qui permet de définir une relation d'ordre $\leq$ sur les diagrammes par inclusion des ensembles abstraits qu'ils représentent.

2.3 Approximations

L'un des objectifs pour cette structure de données est de transformer un diagramme en un autre incluant le diagramme initial et de taille plus faible. Sur les diagrammes de décision abstraits ont été développés deux algorithmes, à partir d'une relation de fusion.

Puisque traiter les diagrammes de manière globale est difficile, on réalise les approximations de manière locale. Plus formellement, une **approximation globale** est une fonction $g : \mathcal{D}_n \to \mathcal{D}_n$ telle que

$$\forall D \in \mathcal{D}_n, D \leq g(D)$$

En pratique, on a surtout développé une **approximation par fusion**, qui est une fonction $f : \mathcal{D}_n \times \mathcal{D}_n \to \mathcal{D}_n$ telle que

$$\begin{cases} \forall A \neq B \in \mathcal{D}_n, A \leq f(A,B) \text{ and } B \leq f(A,B) \\ \forall A \in \mathcal{D}_n, f(A,A) = A \end{cases}$$

Le **théorème de fusion** indique que si l'on dispose d'une approximation par fusion, alors on dispose d'une approximation globale. Il simplifie grandement les preuves ultérieures, puisque l'on peut se contenter de démontrer la propriété d'approximation par fusion pour montrer l'approximation globale.

D'un point de vue computationnel, les approximations par fusion ont aussi l'avantage de pouvoir s'appliquer à des sous-diagrammes : pour réduire un diagramme D, il suffira alors de réaliser une approximation par fusion sur deux sous-diagrammes de D. Il est alors possible de réduire un diagramme de manière locale, ce qui est plus efficace que de considérer le diagramme parent directement.

2.4 Réduction

Deux algorithmes de réduction ont été développés, utilisant largement l'approximation par fusion fm (pour **force merge**) de la figure 2.2, qui est centrale dans la réduction des diagrammes.

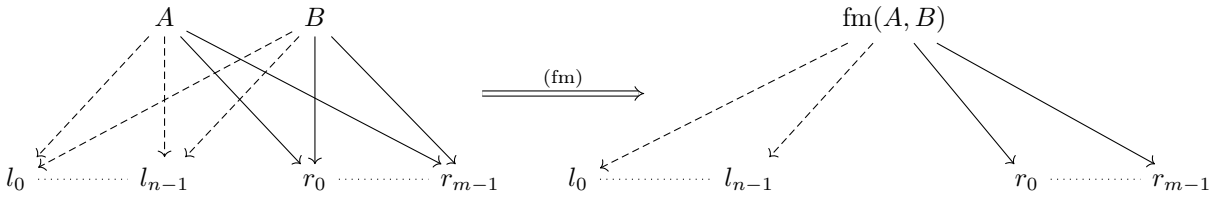

FIGURE 2.2 – Approximation par fusion forcée

où si ampl(A, x) est l'amplitude abstraite sur le lien entre A et x, alors les nouvelles amplitudes abstraites sont définies par la formule suivante

$$\forall x \in \{l_0, ..., l_{k-1}, r_0, ..., r_{m-1}\},\ \text{ampl}(\text{fm}(A,B), x) = \text{ampl}(A,x) \sqcup \text{ampl}(B,x)$$

où $\sqcup$ est l'opération d'union des intervalles complexes (cartésiens ou polaires). Cette approximation par fusion fonctionne en pratique y compris sur des diagrammes n'ayant a priori pas de descendance commune, puisqu'il suffit d'ajouter des branches avec un poids nul pour se ramener à ce cas. On remarque que cette généralisation ne fonctionne que dans le cas où on permet l'utilisation de diagrammes additifs.

2.5 Exemple d'approximation

Considérons le diagramme suivant, où tous les fils (gauche ou droit) sont $\boxed{1}$ et où les branches sans amplitude écrite sont d'amplitude 1. Appliquer fm sur A et B permet fait passer le diagramme additif initial à un diagramme additif abstrait $D' \geq D$.

Ici, la fusion permettrait ensuite de fusionner les deux branches gauches de D' pour obtenir un diagramme non additif, comme le montre la figure 2.3.

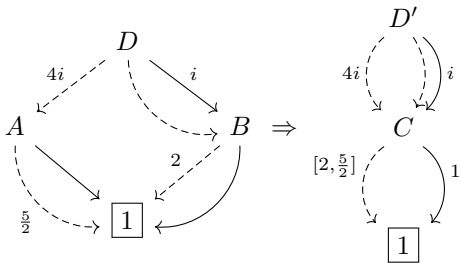

FIGURE 2.3 – Fusion de A et B

2.6 Erreur

S'il est possible de réaliser des fusions sur n'importe quelle paire de diagrammes de même hauteur, savoir lesquels fusionner afin de réduire le diagramme sans causer de trop grande perte de précision est un enjeu majeur. On a donc développé une grandeur d'**erreur**, qui permet de déterminer si une fusion est intéressante ou non. Cette grandeur se distingue en deux intervalles : ρ et ε, définis inductivement sur les diagrammes de hauteur n par

$$\rho(\boxed{1}) = \{1\}$$
$$\varepsilon(\boxed{1}) = \{0\})$$

$$\forall G, D \in \mathscr{P}_f(\mathcal{A}_0 \times \mathcal{D}_n), \rho((G, D)) = \left(\sum_{(l,L)\in G} l\rho(L)\right) \bigsqcup \left(\sum_{(r,R)\in D} r\rho(R)\right)$$

$$\forall G, D \in \mathscr{P}_f(\mathcal{A}_0 \times \mathcal{D}_n),$$
$$\varepsilon((G, D)) = \left(\sum_{(l,L)\in G} l \max|\rho(L) \ominus \varepsilon(L)| + \varepsilon(L)\right) \bigsqcup \left(\sum_{(r,R)\in D} r \max|\rho(R) \ominus \varepsilon(R)| + \varepsilon(R)\right)$$

où α^c est la version centrée d'un intervalle cartésien et où $\ominus$ est l'opération de « rognage » illustrée figure 2.4 et définie lorsque $\alpha \subset \beta$ telle que

$$\forall \alpha, \beta \in \mathcal{A}_0, (\alpha \ominus \beta) + \beta = \alpha$$

Arriver à cette définition pour la grandeur d'erreur a nécessité des recherches conséquentes au cours du semestre. Plusieurs autres définitions ont été mises à l'épreuve, mais celle-ci a été retenue comme la plus prometteuse.

Ceci pourra faire l'objet de recherches ultérieures plus expérimentales, fondées sur des *benchmarks* de réduction de diagrammes. Le choix de cette définition fait respecter à ces grandeurs plusieurs propriétés intéressantes, en particulier ρ contient tous les autres intervalles de l'évaluation

$$\forall D \in \mathcal{D}_n, \forall i \in \{0, ..., 2^n - 1\}, \mathcal{E}(D)[i] \subset \rho(D)$$

ou le fait que l'intervalle ε soit toujours centré. Le calcul de ces grandeurs, s'il est correctement stocké dans la structure de données, n'induit pas de temps de calcul rédhibitoire puisqu'il ne nécessite pas de calculer l'évaluation entière à chaque changement (il suffit de « propager » un changement dans un diagramme fils aux parents).

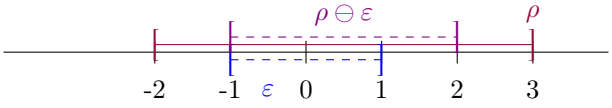

FIGURE 2.4 – Opération de « rognage » $\ominus$ dans le cas des réels

2.7 Application de portes

Il n'existe pas par défaut de formalisation des portes quantiques dans le cas de diagrammes de décision quantiques additifs abstraits, puisque cette structure de données est nouvelle. On a donc défini une application de portes $M \in \mathcal{M}_{2^n,2^n}(\mathbb{C})$ à un diagramme $D \in \mathcal{D}_n$ préservant ce qu'on attend comme effet sur l'évaluation des diagrammes, c'est-à-dire

$$\mathcal{E}(M(D)) = M \cdot \mathcal{E}(D)$$

où $\cdot$ est le produit matriciel basé sur le produit dans $\mathcal{A}_0$. On note que, sans perte de généralité, on peut supposer que les coefficients de M sont eux-mêmes des intervalles complexes. Réalisons un exemple d'application de porte sur un diagramme $D \in \mathcal{D}_n$.

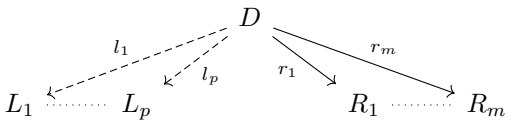

FIGURE 2.5 – Diagramme D

En pratique, pour appliquer une porte M à un diagramme D

1. On sépare M en 4 sous-matrices
$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$$
2. On crée pour chaque branche gauche de D une branche droite de même amplitude et pour chaque branche droite de D une branche gauche de même amplitude.
3. On applique M_{00} à chaque branche gauche (hors celles créées à l'étape 2)
4. On applique M_{01} à chaque branche gauche (uniquement celles créées à l'étape 2)
5. On applique M_{10} à chaque branche droite (uniquement celles créées à l'étape 2)
6. On applique M_{11} à chaque branche droite (hors celles créées à l'étape 2)

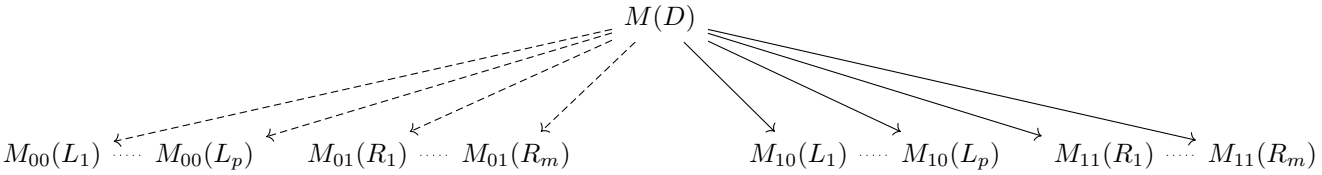

FIGURE 2.6 – Diagramme $M(D)$

L'évaluation de $M(D)$ avec cet algorithme est correcte, puisque comme décrit sur la figure 2.6, on a

$$\mathcal{E}(M(D)) = \begin{pmatrix} \sum l_i M_{00}\mathcal{E}(L_i) + \sum r_j M_{01}\mathcal{E}(R_j) \\ \sum l_i M_{10}\mathcal{E}(L_i) + \sum r_j M_{11}\mathcal{E}(R_j) \end{pmatrix} = M \cdot \mathcal{E}(D)$$

ce qui correspond bien à l'effet attendu de l'application de la porte M sur le diagramme D. Le cas de base (une matrice de taille 1, c'est-à-dire un scalaire) est traité simplement en multipliant les poids des branches par ce scalaire.

On a traité ici le cas d'une porte s'appliquant à tous les qubits. Plus généralement, considérant une porte s'appliquant seulement à k qubits contigus (disons les qubits $Q = q, ..., q+k-1$). Pour un état v à n qubits, appliquer la porte $P \in \mathcal{M}_{2^k,2^k}(\mathbb{C})$ à aux qubits Q de v revient à appliquer à v tout entier la porte

$$M = \left(\bigotimes^{q-1} I\right) \otimes P \otimes \left(\bigotimes^{n-k-q-1} I\right)$$

On remarque donc que l'application d'une porte à un nombre restreint de qubits peut être vue comme l'application d'une porte à tous les qubits, mais avec des matrices identités à la place des matrices de la porte. De plus, utiliser des portes S afin d'échanger la place de qubits permet de traiter avec l'algorithme présenté précédemment tous les cas d'application de portes logiques à un sous-ensemble quelconque des qubits du circuit.

Chapitre 3

Implémentation

L'implémentation a été réalisée en **langage C++**. Ce choix a été motivé par plusieurs raisons : la performance, la maturité du langage et son utilisation dans plusieurs projets historiques du domaine [12] [13], et la proximité avec le langage C, déjà bien connu.

3.1 Structure du code

Le code fait l'objet d'une documentation extensive, disponible en ligne. [9] L'implémentation utilise largement la **programmation orientée objet**, en définissant des classes pour les objets manipulés. Les classes principales sont les suivantes, du plus bas au plus haut niveau :

- Intervalles réels (quelconques, de réels positifs ou modulo 2π), et complexes (cartésiens ou polaires)
- **Diagrammes** (par défaut, additifs et abstraits)
- Les branches, qui contiennent un lien (**pointeur**) vers un digramme de destination et un intervalle complexe (cartésien ou polaire)

Les classes et fonctions sont organisées en **espaces de noms** (*namespaces*) pour éviter les conflits de noms, et définies dans des fichiers séparés. Des fonctions de **réduction** servent ensuite à réduire les diagrammes, en utilisant les fonctions de sélection.

Les définitions des fonctions et méthodes et leur implémentation sont séparées dans des fichiers d'en-tête (*header*) et des fichiers de code source (*source*), respectivement. Afin de tirer parti de la **compilation séparée**, les fichiers d'en-tête sont inclus dans les fichiers de code source, et les fichiers de code source sont compilés en bibliothèques statiques.

Au cours de ce semestre, un effort a été fait pour améliorer l'**interchangeabilité** entre les intervalles cartésiens et polaires, et pour rendre le code plus générique, par exemple en définissant des types adaptés et optimisés pour les matrices de portes ou les états (contenant des complexes ou des intervalles).

3.2 Tests

Afin de garantir la qualité du code, des **tests unitaires** ont été écrits pour la plupart des fonctions et méthodes. Ces tests sont écrits en utilisant la bibliothèque open-source *Google Test*, et sont organisés en **suites de tests** pour chaque classe. [14] Les 46 tests, qui s'exécutent en moins d'une demi-seconde et comptent plusieurs milliers d'assertions, permettent de valider le code et de détecter les bugs ou régressions.

3.3 Interpréteur QASM

Un **interpréteur** pour le langage Open QASM a été réalisé au cours de ce semestre. Il permet de lire un fichier QASM définissant des qubits et y appliquant des portes logiques, d'en générer un diagramme et d'y faire les modifications correspondant aux portes qu'on souhaite lui appliquer, et d'en afficher l'évaluation.

Cet interpréteur peut aussi fonctionner en mode interactif, c'est-à-dire en lisant les instructions depuis l'entrée standard. Il ne supporte pas l'ensemble des instructions QASM, seulement les portes logiques de base (X, H, CX, S, portes de phase). Toutefois, le *back-end* de cet interpréteur est capable d'appliquer n'importe quelle porte grâce à la méthode détaillée en section 2.7. Cet interpréteur est constitué d'une bibliothèque qu'on peut inclure avec l'en-tête `qasm.h` comportant un *namespace* `qasm` et d'un exécutable `prompt`.

Le QASM étant déjà un langage de description de circuits quantiques, il n'a pas été jugé nécessaire de réaliser véritable compilateur comportant un front-end, un middle-end réalisant des optimisations et un back-end

transformant la représentation intermédiaire en exécutable. L'interpréteur a été testé avec succès sur plusieurs exemples de circuits quantiques.

L'exemple du programme 1.2 s'exécute en 2,6 ms. Des **benchmarks** sur un plus grand nombre de qubits, ou sur des circuits plus complexes, n'ont pas été réalisés. De plus il est probable qu'une partie non négligeable du temps d'exécution soit due à la lecture du fichier ou à l'interprétation du QASM, et non à la génération du diagramme et à sa modification.

3.4 Outils

Le code source est versionné à l'aide du logiciel *Git*, et est disponible sur la plateforme *GitHub*. [15] [16] [8] Ce projet fait l'objet d'une **intégration continue** utilisant *GitHub Actions*, automatisant la validation des tests.

Ce semestre a vu l'arrivée de l'utilisation de *Clang* comme compilateur à la place de *GCC*, et de *Ninja* à la place de *Make*. [17] [18] Ces changements ont permis de réduire le temps de compilation, et d'améliorer la lisibilité des messages d'erreur, ainsi que de faciliter l'utilisation de fonctionnalités récentes du langage (datant de C++23). Ces outils sont orchestrés par *CMake*. [19]

Compiler le projet (tests exclus) prend environ 8 secondes sur un ordinateur portable récent. Il faut 5 secondes supplémentaires pour compiler les tests. Le projet compte environ 5 000 lignes de code, dont environ 1 000 lignes de tests.

Le projet fait aussi l'objet d'une **documentation**, écrite dans les commentaires des fichiers d'en-tête. Les pages web de documentation sont générées automatiquement par *Doxygen*. [20] et publiée automatiquement sur *GitHub Pages* à chaque modification à l'aide de *GitHub Actions*. [9]

Chapitre 4

Conclusion

4.1 Travail réalisé

Le modèle qui a été développé, avec algorithme de réduction, permet de limiter autant que souhaité la taille d'un état en mémoire, au prix d'une perte de précision. Le cadre mathématique de celui-ci a été défini, et des algorithmes de réduction ont étés prouvés sur cette structure de données.

L'implémentation, n'a pas encore fourni de résultats expérimentaux significatifs, mais permet déjà de simuler des diagrammes de décision quantiques de manière performante. Sa robustesse est assurée par des tests unitaires.

Au cours de ce semestre, des efforts sur l'implémentation ont été réalisés, entre autres afin de rendre interchangables les intervalles complexes cartésiens et polaires mais aussi plus généralement pour améliorer la qualité et réusabilité du code. La documentation du code a été améliorée, et le document principal détaillant les aspects théoriques du projet a été complété.

Un interpréteur QASM a été réalisé au cours de ce semestre, se reposant sur des travaux théoriques ayant permis une implémentation de l'application de portes à des diagrammes.

Le travail réalisé sur cette période est donc prometteur, et s'incrit dans la continuité du travail réalisé au semestre dernier : malgré des difficultés rencontrées, tant d'un point de vue technique en programmation que dans la définition du modèle, ayant parfois amené à revenir en arrière avant de trouver une bonne définition de l'erreur par exemple, les résultats théoriques obtenus sont prometteurs.

4.2 Perspectives

Plusieurs perspectives peuvent être envisagées pour prolonger les travaux réalisés. D'une part, puisque plusieurs choix dans la définition du modèle ont étés faits de manière arbitraire parmi plusieurs options possibles, il est pertinent de réaliser de nombreuses simulations dans des configurations différentes afin de déterminer les choix les plus pertinents.

On pourra notamment réaliser des tests de performance sur plusieurs fonctions d'erreur, ou observer l'effet d'échanges de qubits sur les performances en termes de manque de précision. De manière générale, il manque des **résultats expérimentaux** et en particulier des comparaisons à d'autres implémentations de la littérature sur des exemples usuels.

Nous pourrons aussi étendre le modèle avec d'autres concepts, comme les automates d'arbres pour la **vérification** ou les diagrammes de décisions et applications localement inversibles. [21] [22] Enfin, l'implémentation aurait à gagner d'être plus facile d'utilisation, par exemple avec une **interface graphique** pouvant agrémenter l'interpréteur QASM. Ces axes de travail pourront être explorés au cours de la suite du projet s'il est prolongé.

Bibliographie

[1] D. Deutsch et R. Jozsa. « Rapid solution of problems by quantum computation ». In : *Proceedings of the Royal Society of London. Series A : Mathematical and Physical Sciences* 439 (1907 1992), p. 553-558. doi : 10.1098/rspa.1992.0167.

[2] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv : quant-ph/9605043 [quant-ph].

[3] Peter W. Shor. « Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer ». In : *SIAM Journal on Computing* 26.5 (oct. 1997), p. 1484-1509. issn : 1095-7111. doi : 10.1137/s0097539795293172.

[4] Andrew W. Cross et al. *Open Quantum Assembly Language*. 2017. arXiv : 1707.03429 [quant-ph].

[5] Shin-ichi Minato. *Binary decision diagrams and applications for VLSI CAD*. eng. Kluwer international series in engineering and computer science. VLSI, computer architecture, and digital signal processing. Springer New York, NY, 1996, p. 7-11. isbn : 1-4612-8558-5.

[6] P. Cousot et R. Cousot. « Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints ». In : *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Los Angeles, California : ACM Press, New York, NY, 1977, p. 238-252.

[7] Mads Rosendahl. *Introduction to abstract interpretation*. CS University of Copenhagen, 1995.

[8] Malo Leroy. *Abstract additive quantum decision diagrams*. Fév. 2025. url : https://github.com/Firefnix/coto/.

[9] Malo Leroy. *Coto Documentation*. Fév. 2025. url : https://firefnix.github.io/coto/.

[10] Teruo Sunaga. « Theory of an interval algebra and its application to numerical analysis [Reprint of Res. Assoc. Appl. Geom. Mem. 2 (1958), 29–46] ». In : *Japan Journal of Industrial and Applied Mathematics* 26.2-3 (2009), p. 125-143.

[11] J. Rokne et P. Lancaster. « Complex interval arithmetic ». In : *Commun. ACM* 14.2 (fév. 1971), p. 111-112. issn : 0001-0782. doi : 10.1145/362515.362563.

[12] Benjamin Bichsel et al. « Abstraqt : Analysis of Quantum Circuits via Abstract Stabilizer Simulation ». In : *Quantum* 7 (nov. 2023), p. 1185. issn : 2521-327X. doi : 10.22331/q-2023-11-20-1185.

[13] Jean-Baptiste Debize Martin Olivier et Théo Fourcat. *QTranslator – Assembly to quantum assembly*. Juin 2022. url : https://github.com/PoCInnovation/QTranslator.

[14] Google. *Google Test v1.16.0*. Fév. 2025. url : https://google.github.io/googletest/.

[15] Git. *Git v2.47.1*. 2005-2025. url : https://git-scm.com/.

[16] GitHub. *GitHub*. 2008-2025. url : https://github.com/.

[17] Apple Inc. *Clang v19.1.7*. 2007-2025. url : https://clang.llvm.org/.

[18] Evan Martin. *Ninja v1.12.1*. 2012-2024. url : https://ninja-build.org/.

[19] Kitware. *CMake v3.31.5*. 2000-2025. url : https://gitlab.kitware.com/cmake/cmake.

[20] Doxygen Dimitri van Heesch. *Doxygen v1.13.2*. 1997-2024. url : https://www.doxygen.nl.

[21] Yu-Fang Chen et al. *An Automata-based Framework for Verification and Bug Hunting in Quantum Circuits (Technical Report)*. 2023. arXiv : 2301.07747 [cs.LO].

[22] Lieuwe Vinkhuijzen et al. « LIMDD : A Decision Diagram for Simulation of Quantum Computing Including Stabilizer States ». In : *Quantum* 7 (sept. 2023), p. 1108. issn : 2521-327X. doi : 10.22331/q-2023-09-11-1108.